

LLNL Nanosecond Gated Camera

2.1.2

Generated by Doxygen 1.10.0

1 Namespace Index	1
1.1 Package List	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Namespace Documentation	9
5.1 nsCamera Namespace Reference	9
5.1.1 Detailed Description	9
5.1.2 Variable Documentation	10
5.1.2.1 __all__	10
5.2 nsCamera.boards Namespace Reference	10
5.2.1 Detailed Description	10
5.2.2 Variable Documentation	10
5.2.2.1 __all__	10
5.3 nsCamera.boards.LLNL_v1 Namespace Reference	11
5.3.1 Detailed Description	11
5.4 nsCamera.boards.LLNL_v4 Namespace Reference	11
5.4.1 Detailed Description	11
5.5 nsCamera.CameraAssembler Namespace Reference	11
5.5.1 Detailed Description	12
5.6 nsCamera.comms Namespace Reference	12
5.6.1 Detailed Description	12
5.6.2 Variable Documentation	13
5.6.2.1 __all__	13
5.7 nsCamera.comms.GigE Namespace Reference	13
5.7.1 Detailed Description	13
5.8 nsCamera.comms.RS422 Namespace Reference	13
5.8.1 Detailed Description	13
5.9 nsCamera.sensors Namespace Reference	14
5.9.1 Detailed Description	14
5.9.2 Variable Documentation	14
5.9.2.1 __all__	14
5.10 nsCamera.sensors.daedalus Namespace Reference	14
5.10.1 Detailed Description	15

5.11 nsCamera.sensors.icarus Namespace Reference	15
5.11.1 Detailed Description	15
5.12 nsCamera.sensors.icarus2 Namespace Reference	15
5.12.1 Detailed Description	16
5.13 nsCamera.sensors.sensorBase Namespace Reference	16
5.13.1 Detailed Description	16
5.14 nsCamera.utils Namespace Reference	16
5.14.1 Detailed Description	17
5.14.2 Variable Documentation	17
5.14.2.1 __all__	17
5.15 nsCamera.utils.crc16pure Namespace Reference	17
5.15.1 Detailed Description	17
5.15.2 Function Documentation	18
5.15.2.1 _crc16()	18
5.15.2.2 crc16xmodem()	18
5.15.3 Variable Documentation	18
5.15.3.1 CRC16_XMODEM_TABLE	18
5.16 nsCamera.utils.FlatField Namespace Reference	19
5.16.1 Detailed Description	19
5.16.2 Function Documentation	20
5.16.2.1 generateFF()	20
5.16.2.2 getFilenames()	20
5.16.2.3 getROIvector()	21
5.16.2.4 removeFF()	21
5.16.2.5 removeFFall()	21
5.16.2.6 tslopes()	22
5.16.3 Variable Documentation	22
5.16.3.1 action	22
5.16.3.2 args	22
5.16.3.3 default	22
5.16.3.4 dest	23
5.16.3.5 directory	23
5.16.3.6 framelist	23
5.16.3.7 help	23
5.16.3.8 nargs	23
5.16.3.9 parser	23
5.17 nsCamera.utils.misc Namespace Reference	23
5.17.1 Detailed Description	24
5.17.2 Function Documentation	24

5.17.2.1 bytes2str()	24
5.17.2.2 checkCRC()	25
5.17.2.3 flattenlist()	25
5.17.2.4 generateFrames()	26
5.17.2.5 getEnter()	27
5.17.2.6 loadDumpedData()	27
5.17.2.7 makeLogLabels()	30
5.17.2.8 partition()	31
5.17.2.9 plotFrames()	32
5.17.2.10 saveTiffs()	33
5.17.2.11 str2bytes()	35
5.17.2.12 str2nparray()	36
5.18 nsCamera.utils.Packet Namespace Reference	36
5.18.1 Detailed Description	36
5.19 nsCamera.utils.Subregister Namespace Reference	37
5.19.1 Detailed Description	37
6 Class Documentation	39
6.1 nsCamera.CameraAssembler.CameraAssembler Class Reference	39
6.1.1 Detailed Description	42
6.1.2 Constructor & Destructor Documentation	43
6.1.2.1 __init__()	43
6.1.3 Member Function Documentation	46
6.1.3.1 abortReadoff()	46
6.1.3.2 arm()	47
6.1.3.3 batchAcquire()	47
6.1.3.4 bytes2str()	49
6.1.3.5 checkCRC()	49
6.1.3.6 checkRegSet()	49
6.1.3.7 checkSensorVoltStat()	50
6.1.3.8 checkStatus()	50
6.1.3.9 checkStatus2()	50
6.1.3.10 clearStatus()	50
6.1.3.11 closeDevice()	51
6.1.3.12 configADCs()	51
6.1.3.13 disarm()	51
6.1.3.14 dumpNumpy()	51
6.1.3.15 dumpRegisters()	52
6.1.3.16 dumpStatus()	53

6.1.3.17 dumpSubregisters()	53
6.1.3.18 enableLED()	54
6.1.3.19 flattenlist()	54
6.1.3.20 getBoardInfo()	54
6.1.3.21 getEnter()	55
6.1.3.22 getManualTiming()	55
6.1.3.23 getMonV()	55
6.1.3.24 getPot()	56
6.1.3.25 getPotV()	58
6.1.3.26 getPressure()	58
6.1.3.27 getRegister()	59
6.1.3.28 getSensTemp()	60
6.1.3.29 getSubregister()	60
6.1.3.30 getTemp()	61
6.1.3.31 getTimer()	61
6.1.3.32 getTiming()	61
6.1.3.33 initBoard()	61
6.1.3.34 initialize()	62
6.1.3.35 initPots()	63
6.1.3.36 initPowerCheck()	63
6.1.3.37 initSensor()	64
6.1.3.38 latchPots()	64
6.1.3.39 mmReadoff()	64
6.1.3.40 parseReadoff()	65
6.1.3.41 partition()	65
6.1.3.42 plotFrames()	65
6.1.3.43 powerCheck()	66
6.1.3.44 printBoardInfo()	66
6.1.3.45 readFrames()	67
6.1.3.46 readImgs()	67
6.1.3.47 readoff()	68
6.1.3.48 readSerial()	68
6.1.3.49 readSRAM()	68
6.1.3.50 reboot()	68
6.1.3.51 reinitialize()	69
6.1.3.52 reportEdgeDetects()	69
6.1.3.53 reportStatus()	69
6.1.3.54 resetTimer()	69
6.1.3.55 resolveSubreg()	70

6.1.3.56 saveFrames()	70
6.1.3.57 saveHDF()	71
6.1.3.58 saveNumpys()	72
6.1.3.59 saveTiffs()	74
6.1.3.60 selectOscillator()	74
6.1.3.61 sendCMD()	74
6.1.3.62 sensorSpecific()	74
6.1.3.63 setArbTiming()	75
6.1.3.64 setExtClk()	75
6.1.3.65 setFrames()	75
6.1.3.66 setHighFullWell()	76
6.1.3.67 setInterlacing()	76
6.1.3.68 setLED()	77
6.1.3.69 setManualShutters()	77
6.1.3.70 setManualTiming()	77
6.1.3.71 setPhiDelay()	77
6.1.3.72 setPot()	78
6.1.3.73 setPotV()	79
6.1.3.74 setPowerSave()	82
6.1.3.75 setPPER()	82
6.1.3.76 setRegister()	83
6.1.3.77 setRows()	84
6.1.3.78 setSubregister()	85
6.1.3.79 setTiming()	86
6.1.3.80 setTriggerDelay()	87
6.1.3.81 setZeroDeadTime()	87
6.1.3.82 startCapture()	87
6.1.3.83 str2bytes()	87
6.1.3.84 str2npararray()	87
6.1.3.85 submitMessages()	88
6.1.3.86 waitForSRAM()	88
6.1.3.87 writeSerial()	89
6.1.4 Member Data Documentation	89
6.1.4.1 abort	89
6.1.4.2 arch	89
6.1.4.3 armed	89
6.1.4.4 board	89
6.1.4.5 boardname	89
6.1.4.6 commname	90

6.1.4.7 comms	90
6.1.4.8 currttime	90
6.1.4.9 cycle	90
6.1.4.10 FPGAbordtype	90
6.1.4.11 FPGAinterfaces	90
6.1.4.12 FPGAinvalid	90
6.1.4.13 FPGANum	91
6.1.4.14 FPGArad	91
6.1.4.15 FPGAsensor	91
6.1.4.16 FPGAVersion	91
6.1.4.17 inittime	91
6.1.4.18 iplist	91
6.1.4.19 logcrit	91
6.1.4.20 logcritbase	92
6.1.4.21 logdebug	92
6.1.4.22 logdebugbase	92
6.1.4.23 logerr	92
6.1.4.24 logerrbase	92
6.1.4.25 loginfo	92
6.1.4.26 loginfobase	92
6.1.4.27 logtag	93
6.1.4.28 logwarn	93
6.1.4.29 logwarnbase	93
6.1.4.30 oldtime	93
6.1.4.31 packageroot	93
6.1.4.32 padToFull	93
6.1.4.33 parsedtime	93
6.1.4.34 payloaderror	94
6.1.4.35 platform	94
6.1.4.36 port	94
6.1.4.37 PY3	94
6.1.4.38 pyth1	94
6.1.4.39 pyth2	94
6.1.4.40 python	94
6.1.4.41 read	95
6.1.4.42 savetime	95
6.1.4.43 sensmanual	95
6.1.4.44 sensor	95
6.1.4.45 sensorname	95

6.1.4.46 sens timing	95
6.1.4.47 timeout	95
6.1.4.48 trigtime	96
6.1.4.49 unstringed	96
6.1.4.50 verblevel	96
6.1.4.51 verbmap	96
6.1.4.52 verbose	96
6.1.4.53 version	96
6.1.4.54 waited	96
6.2 nsCamera.sensors.daedalus.daedalus Class Reference	97
6.2.1 Detailed Description	98
6.2.2 Constructor & Destructor Documentation	99
6.2.2.1 __init__()	99
6.2.3 Member Function Documentation	100
6.2.3.1 getManualTiming()	100
6.2.3.2 getSensTemp()	101
6.2.3.3 parseReadoff()	102
6.2.3.4 reportStatusSensor()	103
6.2.3.5 selectOscillator()	104
6.2.3.6 sensorSpecific()	105
6.2.3.7 setExtClk()	106
6.2.3.8 setHighFullWell()	107
6.2.3.9 setInterlacing()	108
6.2.3.10 setManualShutters()	109
6.2.3.11 setManualTiming()	109
6.2.3.12 setPhiDelay()	110
6.2.3.13 setTriggerDelay()	111
6.2.3.14 setZeroDeadTime()	112
6.2.4 Member Data Documentation	113
6.2.4.1 bytesperpixel	113
6.2.4.2 ca	114
6.2.4.3 columns [1/2]	114
6.2.4.4 columns [2/2]	114
6.2.4.5 detect	114
6.2.4.6 firstframe	114
6.2.4.7 firstrow	114
6.2.4.8 fpganumID	114
6.2.4.9 height	115
6.2.4.10 HFW [1/2]	115

6.2.4.11 HFW [2/2]	115
6.2.4.12 interlacing [1/2]	115
6.2.4.13 interlacing [2/2]	115
6.2.4.14 lastframe	115
6.2.4.15 lastrow	115
6.2.4.16 loglabel	116
6.2.4.17 maxframe	116
6.2.4.18 maxheight	116
6.2.4.19 maxwidth	116
6.2.4.20 minframe	116
6.2.4.21 nframes	116
6.2.4.22 padToFull	116
6.2.4.23 sens_registers	117
6.2.4.24 sens_subregisters	117
6.2.4.25 sensfam	117
6.2.4.26 specwarn	117
6.2.4.27 toffset	117
6.2.4.28 tslope	117
6.2.4.29 width	117
6.2.4.30 ZDT [1/2]	118
6.2.4.31 ZDT [2/2]	118
6.3 nsCamera.utils.misc.fakeCA Class Reference	118
6.3.1 Detailed Description	119
6.3.2 Constructor & Destructor Documentation	119
6.3.2.1 __init__()	119
6.3.3 Member Function Documentation	120
6.3.3.1 partition()	120
6.3.4 Member Data Documentation	120
6.3.4.1 boardname	120
6.3.4.2 logcrit	120
6.3.4.3 logcritbase	120
6.3.4.4 logdebug	120
6.3.4.5 logdebugbase	120
6.3.4.6 logerr	121
6.3.4.7 logerrbase	121
6.3.4.8 loginfo	121
6.3.4.9 loginfobase	121
6.3.4.10 logtag	121
6.3.4.11 logwarn	121

6.3.4.12 logwarnbase	121
6.3.4.13 padToFull	122
6.3.4.14 sensor	122
6.3.4.15 sensorname	122
6.4 nsCamera.comms.GigE.GigE Class Reference	122
6.4.1 Detailed Description	123
6.4.2 Constructor & Destructor Documentation	124
6.4.2.1 __init__()	124
6.4.3 Member Function Documentation	125
6.4.3.1 arm()	125
6.4.3.2 closeDevice()	126
6.4.3.3 getCardInfo()	127
6.4.3.4 getCardIP()	127
6.4.3.5 openDevice()	127
6.4.3.6 readFrames()	128
6.4.3.7 readoff()	129
6.4.3.8 readSerial()	130
6.4.3.9 sendCMD()	131
6.4.3.10 writeSerial()	132
6.4.4 Member Data Documentation	133
6.4.4.1 _zest	133
6.4.4.2 ca	133
6.4.4.3 CardInfo	133
6.4.4.4 CardInfoP	134
6.4.4.5 closecard	134
6.4.4.6 Connection	134
6.4.4.7 dport	134
6.4.4.8 logcrit	134
6.4.4.9 logdebug	134
6.4.4.10 logerr	134
6.4.4.11 loginfo	135
6.4.4.12 logwarn	135
6.4.4.13 mode	135
6.4.4.14 payloadsize	135
6.4.4.15 readTimeout	135
6.4.4.16 skipError	135
6.4.4.17 writeTimeout	135
6.4.4.18 ZCountCards	136
6.4.4.19 ZErrorDict	136

6.4.4.20 ZESTETM1_CARD_INFO	136
6.4.4.21 ZOpenConnection	136
6.4.4.22 ZReadData	136
6.4.4.23 ZWriteData	136
6.5 nsCamera.sensors.icarus.icarus Class Reference	137
6.5.1 Detailed Description	138
6.5.2 Constructor & Destructor Documentation	139
6.5.2.1 __init__()	139
6.5.3 Member Function Documentation	140
6.5.3.1 checkSensorVoltStat()	140
6.5.3.2 sensorSpecific()	140
6.5.4 Member Data Documentation	141
6.5.4.1 bytesperpixel	141
6.5.4.2 ca	141
6.5.4.3 columns	141
6.5.4.4 detect	141
6.5.4.5 firstframe	141
6.5.4.6 firstrow	142
6.5.4.7 fpganumID	142
6.5.4.8 height	142
6.5.4.9 icarustype	142
6.5.4.10 interlacing	142
6.5.4.11 lastframe	142
6.5.4.12 lastrow	142
6.5.4.13 loglabel	143
6.5.4.14 maxframe	143
6.5.4.15 maxheight	143
6.5.4.16 maxwidth	143
6.5.4.17 minframe	143
6.5.4.18 nframes	143
6.5.4.19 padToFull	143
6.5.4.20 sens_registers	144
6.5.4.21 sens_subregisters	144
6.5.4.22 sensfam	144
6.5.4.23 specwarn	144
6.5.4.24 width	144
6.6 nsCamera.sensors.icarus2.icarus2 Class Reference	144
6.6.1 Detailed Description	146
6.6.2 Constructor & Destructor Documentation	146

6.6.2.1 <code>__init__()</code>	146
6.6.3 Member Function Documentation	147
6.6.3.1 <code>sensorSpecific()</code>	147
6.6.4 Member Data Documentation	148
6.6.4.1 <code>bytesperpixel</code>	148
6.6.4.2 <code>ca</code>	148
6.6.4.3 <code>columns</code>	148
6.6.4.4 <code>detect</code>	148
6.6.4.5 <code>firstframe</code>	148
6.6.4.6 <code>firstrow</code>	149
6.6.4.7 <code>fpganumID</code>	149
6.6.4.8 <code>height</code>	149
6.6.4.9 <code>icarustype</code>	149
6.6.4.10 <code>interlacing</code>	149
6.6.4.11 <code>lastframe</code>	149
6.6.4.12 <code>lastrow</code>	149
6.6.4.13 <code>loglabel</code>	150
6.6.4.14 <code>maxframe</code>	150
6.6.4.15 <code>maxheight</code>	150
6.6.4.16 <code>maxwidth</code>	150
6.6.4.17 <code>minframe</code>	150
6.6.4.18 <code>nframes</code>	150
6.6.4.19 <code>padToFull</code>	150
6.6.4.20 <code>sens_registers</code>	151
6.6.4.21 <code>sens_subregisters</code>	151
6.6.4.22 <code>sensfam</code>	151
6.6.4.23 <code>specwarn</code>	151
6.6.4.24 <code>width</code>	151
6.7 <code>nsCamera.boards.LLNL_v1.llnl_v1</code> Class Reference	152
6.7.1 Detailed Description	153
6.7.2 Constructor & Destructor Documentation	153
6.7.2.1 <code>__init__()</code>	153
6.7.3 Member Function Documentation	155
6.7.3.1 <code>checkStatus()</code>	155
6.7.3.2 <code>checkStatus2()</code>	155
6.7.3.3 <code>clearStatus()</code>	156
6.7.3.4 <code>configADCs()</code>	156
6.7.3.5 <code>disarm()</code>	157
6.7.3.6 <code>dumpStatus()</code>	157

6.7.3.7 enableLED()	158
6.7.3.8 getPressure()	159
6.7.3.9 getTemp()	159
6.7.3.10 getTimer()	160
6.7.3.11 initBoard()	160
6.7.3.12 initPots()	161
6.7.3.13 initSensor()	162
6.7.3.14 latchPots()	162
6.7.3.15 readSRAM()	163
6.7.3.16 reportEdgeDetects()	163
6.7.3.17 reportStatus()	164
6.7.3.18 resetTimer()	165
6.7.3.19 setLED()	165
6.7.3.20 setPowerSave()	166
6.7.3.21 setPPER()	166
6.7.3.22 softReboot()	167
6.7.3.23 startCapture()	167
6.7.3.24 waitForSRAM()	168
6.7.4 Member Data Documentation	169
6.7.4.1 ADC5_bipolar	169
6.7.4.2 ADC5_mult	169
6.7.4.3 ca	169
6.7.4.4 daedalus_monitor_controls	169
6.7.4.5 daedalus_subreg_aliases	169
6.7.4.6 icarus_monitor_controls	169
6.7.4.7 icarus_subreg_aliases	169
6.7.4.8 logcrit	170
6.7.4.9 logdebug	170
6.7.4.10 logerr	170
6.7.4.11 loginfo	170
6.7.4.12 logwarn	170
6.7.4.13 registers	170
6.7.4.14 rs422_baud	170
6.7.4.15 rs422_cmd_wait	171
6.7.4.16 subregisters	171
6.7.4.17 subreglist	171
6.7.4.18 VREF	171
6.8 nsCamera.boards.LLNL_v4.Llnl_v4 Class Reference	171
6.8.1 Detailed Description	172

6.8.2 Constructor & Destructor Documentation	173
6.8.2.1 <code>__init__()</code>	173
6.8.3 Member Function Documentation	175
6.8.3.1 <code>checkStatus()</code>	175
6.8.3.2 <code>checkStatus2()</code>	175
6.8.3.3 <code>clearStatus()</code>	176
6.8.3.4 <code>configADCs()</code>	176
6.8.3.5 <code>disarm()</code>	177
6.8.3.6 <code>dumpStatus()</code>	177
6.8.3.7 <code>enableLED()</code>	178
6.8.3.8 <code>getPressure()</code>	179
6.8.3.9 <code>getTemp()</code>	180
6.8.3.10 <code>getTimer()</code>	180
6.8.3.11 <code>initBoard()</code>	181
6.8.3.12 <code>initPots()</code>	181
6.8.3.13 <code>initSensor()</code>	182
6.8.3.14 <code>latchPots()</code>	182
6.8.3.15 <code>readSRAM()</code>	183
6.8.3.16 <code>reportEdgeDetects()</code>	183
6.8.3.17 <code>reportStatus()</code>	184
6.8.3.18 <code>resetTimer()</code>	185
6.8.3.19 <code>setLED()</code>	185
6.8.3.20 <code>setPowerSave()</code>	186
6.8.3.21 <code>setPPER()</code>	186
6.8.3.22 <code>softReboot()</code>	187
6.8.3.23 <code>startCapture()</code>	187
6.8.3.24 <code>waitForSRAM()</code>	188
6.8.4 Member Data Documentation	189
6.8.4.1 <code>ADC5_bipolar</code>	189
6.8.4.2 <code>ADC5_mult</code>	189
6.8.4.3 <code>ca</code>	189
6.8.4.4 <code>daedalus_monitor_controls</code>	189
6.8.4.5 <code>daedalus_subreg_aliases</code>	189
6.8.4.6 <code>defoff</code>	189
6.8.4.7 <code>defsens</code>	189
6.8.4.8 <code>icarus_monitor_controls</code>	190
6.8.4.9 <code>icarus_subreg_aliases</code>	190
6.8.4.10 <code>logcrit</code>	190
6.8.4.11 <code>logdebug</code>	190

6.8.4.12 logerr	190
6.8.4.13 loginfo	190
6.8.4.14 logwarn	190
6.8.4.15 registers	191
6.8.4.16 rs422_baud	191
6.8.4.17 rs422_cmd_wait	191
6.8.4.18 subregisters	191
6.8.4.19 subreglist	191
6.8.4.20 VREF	191
6.9 nsCamera.utils.Packet.Packet Class Reference	192
6.9.1 Detailed Description	192
6.9.2 Constructor & Destructor Documentation	193
6.9.2.1 __init__()	193
6.9.3 Member Function Documentation	193
6.9.3.1 calculateCRC()	193
6.9.3.2 checkCRC()	194
6.9.3.3 checkReadPacket()	194
6.9.3.4 checkResponsePacket()	195
6.9.3.5 checkResponseString()	195
6.9.3.6 pktStr()	196
6.9.4 Member Data Documentation	196
6.9.4.1 addr	196
6.9.4.2 cmd	197
6.9.4.3 crc	197
6.9.4.4 data	197
6.9.4.5 payload	197
6.9.4.6 payload_length	197
6.9.4.7 preamble	197
6.9.4.8 PY3	197
6.9.4.9 seqID	198
6.9.4.10 type	198
6.10 nsCamera.comms.RS422.RS422 Class Reference	198
6.10.1 Detailed Description	199
6.10.2 Constructor & Destructor Documentation	199
6.10.2.1 __init__()	199
6.10.3 Member Function Documentation	201
6.10.3.1 arm()	201
6.10.3.2 closeDevice()	202
6.10.3.3 readFrames()	202

6.10.3.4 readoff()	203
6.10.3.5 readSerial()	204
6.10.3.6 sendCMD()	205
6.10.3.7 serialClose()	207
6.10.3.8 writeSerial()	207
6.10.4 Member Data Documentation	208
6.10.4.1 _ser	208
6.10.4.2 baud	208
6.10.4.3 ca	208
6.10.4.4 datatimeout	209
6.10.4.5 logcrit	209
6.10.4.6 logdebug	209
6.10.4.7 logerr	209
6.10.4.8 loginfo	209
6.10.4.9 logwarn	209
6.10.4.10 mode	209
6.10.4.11 par	210
6.10.4.12 payloadsize	210
6.10.4.13 port	210
6.10.4.14 PY3	210
6.10.4.15 read_timeout	210
6.10.4.16 skipError	210
6.10.4.17 stop	210
6.10.4.18 write_timeout	211
6.11 nsCamera.sensors.sensorBase.sensorBase Class Reference	211
6.11.1 Detailed Description	212
6.11.2 Constructor & Destructor Documentation	212
6.11.2.1 __init__()	212
6.11.3 Member Function Documentation	213
6.11.3.1 checkSensorVoltStat()	213
6.11.3.2 getManualTiming()	213
6.11.3.3 getSensorStatus()	214
6.11.3.4 getSensTemp()	214
6.11.3.5 getTiming()	215
6.11.3.6 init_board_specific()	216
6.11.3.7 parseReadoff()	217
6.11.3.8 reportStatusSensor()	217
6.11.3.9 selectOscillator()	218
6.11.3.10 setArbTiming()	218

6.11.3.11 setExtClk()	220
6.11.3.12 setHighFullWell()	221
6.11.3.13 setInterlacing()	221
6.11.3.14 setManualShutters()	222
6.11.3.15 setManualTiming()	222
6.11.3.16 setPhiDelay()	223
6.11.3.17 setTiming()	224
6.11.3.18 setTriggerDelay()	226
6.11.3.19 setZeroDeadTime()	227
6.11.4 Member Data Documentation	227
6.11.4.1 ca	227
6.11.4.2 fpganumID	227
6.11.4.3 logcrit	227
6.11.4.4 logdebug	227
6.11.4.5 logerr	228
6.11.4.6 loginfo	228
6.11.4.7 logwarn	228
6.11.4.8 sensfam	228
6.12 nsCamera.utils.Subregister.SubRegister Class Reference	228
6.12.1 Detailed Description	229
6.12.2 Constructor & Destructor Documentation	229
6.12.2.1 __init__()	229
6.12.3 Member Data Documentation	229
6.12.3.1 addr	229
6.12.3.2 max	230
6.12.3.3 max_value	230
6.12.3.4 maxV	230
6.12.3.5 min	230
6.12.3.6 minV	230
6.12.3.7 name	230
6.12.3.8 register	230
6.12.3.9 resolution	231
6.12.3.10 start_bit	231
6.12.3.11 value	231
6.12.3.12 width	231
6.12.3.13 writable	231
6.13 nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO Class Reference	231
6.13.1 Detailed Description	232
6.13.2 Member Data Documentation	232

6.13.2.1 <code>_fields_</code>	232
6.13.2.2 <code>ubyte4</code>	232
6.13.2.3 <code>ubyte6</code>	232
7 File Documentation	233
7.1 <code>C:/Users/hill35/git/camera_python/nsCamera/__init__.py</code> File Reference	233
7.2 <code>__init__.py</code>	233
7.3 <code>C:/Users/hill35/git/camera_python/nsCamera/boards/__init__.py</code> File Reference	234
7.4 <code>__init__.py</code>	234
7.5 <code>C:/Users/hill35/git/camera_python/nsCamera/comms/__init__.py</code> File Reference	235
7.6 <code>__init__.py</code>	235
7.7 <code>C:/Users/hill35/git/camera_python/nsCamera/sensors/__init__.py</code> File Reference	235
7.8 <code>__init__.py</code>	236
7.9 <code>C:/Users/hill35/git/camera_python/nsCamera/utils/__init__.py</code> File Reference	236
7.10 <code>__init__.py</code>	237
7.11 <code>C:/Users/hill35/git/camera_python/nsCamera/boards/LLNL_v1.py</code> File Reference	237
7.12 <code>LLNL_v1.py</code>	238
7.13 <code>C:/Users/hill35/git/camera_python/nsCamera/boards/LLNL_v4.py</code> File Reference	249
7.14 <code>LLNL_v4.py</code>	249
7.15 <code>C:/Users/hill35/git/camera_python/nsCamera/CameraAssembler.py</code> File Reference	260
7.16 <code>CameraAssembler.py</code>	261
7.17 <code>C:/Users/hill35/git/camera_python/nsCamera/comms/GigE.py</code> File Reference	287
7.18 <code>GigE.py</code>	287
7.19 <code>C:/Users/hill35/git/camera_python/nsCamera/comms/RS422.py</code> File Reference	294
7.20 <code>RS422.py</code>	294
7.21 <code>C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/Data.c</code> File Reference	301
7.21.1 Class Documentation	302
7.21.1.1 struct <code>ZESTETM1_WRITE_REG_CMD</code>	302
7.21.1.2 struct <code>ZESTETM1_WRITE_REG_RESPONSE</code>	302
7.21.1.3 struct <code>ZESTETM1_READ_REG_CMD</code>	303
7.21.1.4 struct <code>ZESTETM1_READ_REG_RESPONSE</code>	303
7.21.1.5 struct <code>ZESTETM1_MAILBOX_INT_CMD</code>	303
7.21.1.6 struct <code>ZESTETM1_MAILBOX_INT_RESPONSE</code>	303
7.21.2 Macro Definition Documentation	304
7.21.2.1 <code>_CRT_SECURE_NO_WARNINGS</code>	304
7.21.2.2 <code>ZESTETM1_COMMAND_MAILBOX_INT</code>	304
7.21.2.3 <code>ZESTETM1_COMMAND_READ_REG</code>	304
7.21.2.4 <code>ZESTETM1_COMMAND_SPI</code>	304
7.21.2.5 <code>ZESTETM1_COMMAND_WRITE_REG</code>	304

7.21.3 Function Documentation	305
7.21.3.1 ZestETM1_CloseConnection()	305
7.21.3.2 ZestETM1_OpenConnection()	305
7.21.3.3 ZestETM1_ReadData()	307
7.21.3.4 ZestETM1_SendCommand()	308
7.21.3.5 ZestETM1_SPIReadWrite()	309
7.21.3.6 ZestETM1_WriteData()	309
7.21.3.7 ZestETM1CloseConnection()	311
7.21.3.8 ZestETM1OpenConnection()	311
7.21.3.9 ZestETM1ReadData()	312
7.21.3.10 ZestETM1ReadRegister()	312
7.21.3.11 ZestETM1SetInterrupt()	313
7.21.3.12 ZestETM1SPIReadWrite()	313
7.21.3.13 ZestETM1WriteData()	314
7.21.3.14 ZestETM1WriteRegister()	314
7.22 Data.c	315
7.23 C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/Error.c File Reference	326
7.23.1 Function Documentation	326
7.23.1.1 ZestETM1GetErrorMessage()	326
7.23.1.2 ZestETM1RegisterErrorHandler()	326
7.23.2 Variable Documentation	327
7.23.2.1 ZestETM1_ErrorHandler	327
7.23.2.2 ZestETM1_ErrorStrings	327
7.24 Error.c	327
7.25 C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/Main.c File Reference	329
7.25.1 Function Documentation	329
7.25.1.1 ZestETM1Close()	329
7.25.1.2 ZestETM1Init()	329
7.26 Main.c	330
7.27 C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/Private.h File Reference	331
7.27.1 Class Documentation	332
7.27.1.1 struct ZESTETM1_CONNECTION_STRUCT	332
7.27.2 Macro Definition Documentation	333
7.27.2.1 closesocket	333
7.27.2.2 SD_BOTH	333
7.27.2.3 ZESTETM1_CONNECTION_HANDLE_MAGIC	333
7.27.2.4 ZESTETM1_DEFAULT_TIMEOUT	333
7.27.2.5 ZESTETM1_ERROR	333
7.27.2.6 ZESTETM1_ERROR_CONN	334

7.27.2.7 ZESTETM1_ERROR_GENERAL	334
7.27.2.8 ZESTETM1_ERROR_STRING	334
7.27.2.9 ZESTETM1_RATE_10MHz	335
7.27.2.10 ZESTETM1_RATE_20MHz	335
7.27.2.11 ZESTETM1_RATE_40MHz	335
7.27.2.12 ZESTETM1_REVERSE	335
7.27.2.13 ZESTETM1_USER_DEVICE_ID	335
7.27.3 Typedef Documentation	335
7.27.3.1 SOCKET	335
7.27.4 Function Documentation	336
7.27.4.1 ZestETM1_CloseConnection()	336
7.27.4.2 ZestETM1_EraseFlashSector()	336
7.27.4.3 ZestETM1_OpenConnection()	336
7.27.4.4 ZestETM1_SendCommand()	338
7.27.4.5 ZestETM1_SPIReadWrite()	338
7.27.4.6 ZestETM1_WriteFlash()	339
7.27.5 Variable Documentation	340
7.27.5.1 ZestETM1_ErrorHandler	340
7.27.5.2 ZestETM1_ErrorStrings	340
7.28 Private.h	340
7.29 C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/UPnP.c File Reference	342
7.29.1 Class Documentation	343
7.29.1.1 struct ZESTETM1_GET_SETTINGS_CMD	343
7.29.1.2 struct ZESTETM1_GET_SETTINGS_RESPONSE	343
7.29.2 Macro Definition Documentation	344
7.29.2.1 _CRT_SECURE_NO_WARNINGS	344
7.29.2.2 _strnicmp	344
7.29.2.3 ZESTETM1_COMMAND_GET_SETTINGS	344
7.29.2.4 ZESTETM1_UPNP_ADDR	344
7.29.2.5 ZESTETM1_UPNP_PORT	344
7.29.3 Function Documentation	345
7.29.3.1 ZestETM1_GetAllAdapters()	345
7.29.3.2 ZestETM1_GetCardInfo()	346
7.29.3.3 ZestETM1_HTTPGet()	348
7.29.3.4 ZestETM1_JoinGroup()	349
7.29.3.5 ZestETM1_LeaveGroup()	349
7.29.3.6 ZestETM1_ReadSettings()	350
7.29.3.7 ZestETM1_CountCards()	350
7.29.3.8 ZestETM1_FreeCards()	353

7.29.3.9 ZestETM1GetCardInfo()	353
7.29.4 Variable Documentation	353
7.29.4.1 ZestETM1_SearchReq	353
7.30 UPnP.c	354
7.31 C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/ZestETM1.h File Reference	362
7.31.1 Class Documentation	364
7.31.1.1 struct ZESTETM1_CARD_INFO	364
7.31.2 Macro Definition Documentation	364
7.31.2.1 ZESTETM1_ERROR_BASE	364
7.31.2.2 ZESTETM1_INFO_BASE	364
7.31.2.3 ZESTETM1_VERSION_FALLBACK	364
7.31.2.4 ZESTETM1_WARNING_BASE	364
7.31.3 Typedef Documentation	365
7.31.3.1 ZESTETM1_CONNECTION	365
7.31.3.2 ZESTETM1_ERROR_FUNC	365
7.31.3.3 ZESTETM1_HANDLE	365
7.31.4 Enumeration Type Documentation	365
7.31.4.1 ZESTETM1_CONNECTION_TYPE	365
7.31.4.2 ZESTETM1_SPI_RATE	365
7.31.4.3 ZESTETM1_STATUS	366
7.31.5 Function Documentation	367
7.31.5.1 ZestETM1Close()	367
7.31.5.2 ZestETM1CloseConnection()	367
7.31.5.3 ZestETM1CountCards()	367
7.31.5.4 ZestETM1FreeCards()	369
7.31.5.5 ZestETM1GetCardInfo()	369
7.31.5.6 ZestETM1GetErrorMessage()	370
7.31.5.7 ZestETM1Init()	370
7.31.5.8 ZestETM1OpenConnection()	371
7.31.5.9 ZestETM1ReadData()	371
7.31.5.10 ZestETM1ReadRegister()	371
7.31.5.11 ZestETM1RegisterErrorHandler()	372
7.31.5.12 ZestETM1SetInterrupt()	372
7.31.5.13 ZestETM1SPIReadWrite()	373
7.31.5.14 ZestETM1WriteData()	373
7.31.5.15 ZestETM1WriteRegister()	374
7.32 ZestETM1.h	375
7.33 C:/Users/hill35/git/camera_python/nsCamera/sensors/daedalus.py File Reference	377
7.34 daedalus.py	377

7.35 C:/Users/hill35/git/camera_python/nsCamera/sensors/icarus.py File Reference	387
7.36 icarus.py	387
7.37 C:/Users/hill35/git/camera_python/nsCamera/sensors/icarus2.py File Reference	390
7.38 icarus2.py	390
7.39 C:/Users/hill35/git/camera_python/nsCamera/sensors/sensorBase.py File Reference	392
7.40 sensorBase.py	392
7.41 C:/Users/hill35/git/camera_python/nsCamera/utls/crc16pure.py File Reference	401
7.42 crc16pure.py	402
7.43 C:/Users/hill35/git/camera_python/nsCamera/utls/FlatField.py File Reference	405
7.44 FlatField.py	406
7.45 C:/Users/hill35/git/camera_python/nsCamera/utls/misc.py File Reference	409
7.46 misc.py	409
7.47 C:/Users/hill35/git/camera_python/nsCamera/utls/Packet.py File Reference	418
7.48 Packet.py	419
7.49 C:/Users/hill35/git/camera_python/nsCamera/utls/Subregister.py File Reference	421
7.50 Subregister.py	422

Index

423

Chapter 1

Namespace Index

1.1 Package List

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

nsCamera	9
nsCamera.boards	10
nsCamera.boards.LLNL_v1	11
nsCamera.boards.LLNL_v4	11
nsCamera.CameraAssembler	11
nsCamera.comms	12
nsCamera.comms.GigE	13
nsCamera.comms.RS422	13
nsCamera.sensors	14
nsCamera.sensors.daedalus	14
nsCamera.sensors.icarus	15
nsCamera.sensors.icarus2	15
nsCamera.sensors.sensorBase	16
nsCamera.utils	16
nsCamera.utils.crc16pure	17
nsCamera.utils.FlatField	19
nsCamera.utils.misc	23
nsCamera.utils.Packet	36
nsCamera.utils.Subregister	37

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

nsCamera.CameraAssembler.CameraAssembler	39
nsCamera.utils.misc.fakeCA	118
nsCamera.comms.GigE.GigE	122
nsCamera.boards.LLNL_v1.llnl_v1	152
nsCamera.boards.LLNL_v4.llnl_v4	171
object	
nsCamera.sensors.sensorBase.sensorBase	211
nsCamera.sensors.daedalus.daedalus	97
nsCamera.sensors.icarus2.icarus2	144
nsCamera.sensors.icarus.icarus	137
nsCamera.utils.Packet.Packet	192
nsCamera.comms.RS422.RS422	198
C.Structure	
nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO	231
nsCamera.utils.Subregister.SubRegister	228
ZESTETM1_CARD_INFO	362
ZESTETM1_CONNECTION_STRUCT	331
ZESTETM1_GET_SETTINGS_CMD	342
ZESTETM1_GET_SETTINGS_RESPONSE	342
ZESTETM1_MAILBOX_INT_CMD	301
ZESTETM1_MAILBOX_INT_RESPONSE	301
ZESTETM1_READ_REG_CMD	301
ZESTETM1_READ_REG_RESPONSE	301
ZESTETM1_WRITE_REG_CMD	301
ZESTETM1_WRITE_REG_RESPONSE	301

Chapter 3

Class Index

3.1 Class List

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

nsCamera.CameraAssembler.CameraAssembler	39
nsCamera.sensors.daedalus.daedalus	97
nsCamera.utils.misc.fakeCA	118
nsCamera.comms.GigE.GigE	122
nsCamera.sensors.icarus.icarus	137
nsCamera.sensors.icarus2.icarus2	144
nsCamera.boards.LLNL_v1.llnl_v1	152
nsCamera.boards.LLNL_v4.llnl_v4	171
nsCamera.utils.Packet.Packet	192
nsCamera.comms.RS422.RS422	198
nsCamera.sensors.sensorBase.sensorBase	211
nsCamera.utils.Subregister.SubRegister	228
nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO	231

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

C:/Users/hill35/git/camera_python/nsCamera/___init___py	233
C:/Users/hill35/git/camera_python/nsCamera/CameraAssembler.py	260
C:/Users/hill35/git/camera_python/nsCamera/boards/___init___py	234
C:/Users/hill35/git/camera_python/nsCamera/boards/LLNL_v1.py	237
C:/Users/hill35/git/camera_python/nsCamera/boards/LLNL_v4.py	249
C:/Users/hill35/git/camera_python/nsCamera/comms/___init___py	235
C:/Users/hill35/git/camera_python/nsCamera/comms/GigE.py	287
C:/Users/hill35/git/camera_python/nsCamera/comms/RS422.py	294
C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/Data.c	301
C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/Error.c	326
C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/Main.c	329
C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/Private.h	331
C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/UPnP.c	342
C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/ZestETM1.h	362
C:/Users/hill35/git/camera_python/nsCamera/sensors/___init___py	235
C:/Users/hill35/git/camera_python/nsCamera/sensors/daedalus.py	377
C:/Users/hill35/git/camera_python/nsCamera/sensors/icarus.py	387
C:/Users/hill35/git/camera_python/nsCamera/sensors/icarus2.py	390
C:/Users/hill35/git/camera_python/nsCamera/sensors/sensorBase.py	392
C:/Users/hill35/git/camera_python/nsCamera/utills/___init___py	236
C:/Users/hill35/git/camera_python/nsCamera/utills/crc16pure.py	401
C:/Users/hill35/git/camera_python/nsCamera/utills/FlatField.py	405
C:/Users/hill35/git/camera_python/nsCamera/utills/misc.py	409
C:/Users/hill35/git/camera_python/nsCamera/utills/Packet.py	418
C:/Users/hill35/git/camera_python/nsCamera/utills/Subregister.py	421

Chapter 5

Namespace Documentation

5.1 nsCamera Namespace Reference

Namespaces

- namespace [boards](#)
- namespace [CameraAssembler](#)
- namespace [comms](#)
- namespace [sensors](#)
- namespace [utils](#)

Variables

- list [__all__](#) = ["CameraAssembler"]

5.1.1 Detailed Description

Created on Tue Mar 22 15:47:43 2016

The Package includes a Camera object and an assembler.

The camera object will be the workhorse of the API. The assembler is used to create the Camera object.

Author: Matthew Dayton (dayton5@llnl.gov)

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

5.1.2 Variable Documentation

5.1.2.1 `__all__`

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

Definition at line 26 of file `__init__.py`.

5.2 nsCamera.boards Namespace Reference

Namespaces

- namespace `LLNL_v1`
- namespace `LLNL_v4`

Variables

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

5.2.1 Detailed Description

This package is a collection of modules that represent the camera boards. Each board has its own number of ADCs, POTs, and sensors. More devices can be added in the future. The list of imports will grow as we make more types of boards.

Author: Matthew Dayton (dayton5@llnl.gov)

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

5.2.2 Variable Documentation

5.2.2.1 `__all__`

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

Definition at line 23 of file `__init__.py`.

5.3 nsCamera.boards.LLNL_v1 Namespace Reference

Classes

- class [llnl_v1](#)

5.3.1 Detailed Description

LLNLv1 board definition, including monitors, pots, and other board-specific settings

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

Author: Matthew Dayton (dayton5@llnl.gov)

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

5.4 nsCamera.boards.LLNL_v4 Namespace Reference

Classes

- class [llnl_v4](#)

5.4.1 Detailed Description

LLNLv4 board definition, including monitors, DACS, and other board-specific settings

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

Author: Matthew Dayton (dayton5@llnl.gov)

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

5.5 nsCamera.CameraAssembler Namespace Reference

Classes

- class [CameraAssembler](#)

5.5.1 Detailed Description

CameraAssembler assembles the separate camera parts into a camera object. This object controls a combination of three components:

1. board : FPGA board -- LLNL_V1, LLNL_V4
2. comms: communication interface -- GigE, RS422
3. sensor : sensor type -- icarus, icarus2, daedalus

Author: Jeremy Martin Hill (jerhill@llnl.gov)
 Author: Matthew Dayton (dayton5@llnl.gov)

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

5.6 nsCamera.comms Namespace Reference

Namespaces

- namespace [GigE](#)
- namespace [RS422](#)

Variables

- list [__all__](#) = ["RS422", "GigE"]

5.6.1 Detailed Description

Created on Tue Mar 22 15:47:43 2016

This package is a collection of modules for uniform handling of the nsCamera's communication systems

Author: Matthew Dayton (dayton5@llnl.gov)

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

5.6.2 Variable Documentation

5.6.2.1 `__all__`

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

Definition at line 25 of file `__init__.py`.

5.7 nsCamera.comms.GigE Namespace Reference

Classes

- class [GigE](#)

5.7.1 Detailed Description

Gigabit Ethernet interface for nsCamera.

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

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

5.8 nsCamera.comms.RS422 Namespace Reference

Classes

- class [RS422](#)

5.8.1 Detailed Description

RS422 driver for nsCamera

Author: Brad Funsten (funsten1@llnl.gov)

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

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

5.9 nsCamera.sensors Namespace Reference

Namespaces

- namespace [daedalus](#)
- namespace [icarus](#)
- namespace [icarus2](#)
- namespace [sensorBase](#)

Variables

- list [__all__](#) = ["icarus", "icarus2", "daedalus"]

5.9.1 Detailed Description

This package is a collection of modules for uniform handling of the various sensors

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

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

5.9.2 Variable Documentation

5.9.2.1 [__all__](#)

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

Definition at line 23 of file [__init__.py](#).

5.10 nsCamera.sensors.daedalus Namespace Reference

Classes

- class [daedalus](#)

5.10.1 Detailed Description

Parameters and functions specific to the daedalus three-frame sensor

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

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

5.11 nsCamera.sensors.icarus Namespace Reference

Classes

- class [icarus](#)

5.11.1 Detailed Description

Parameters and functions specific to the icarus two-frame sensor

Do not use this file as a template for new code development

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

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

5.12 nsCamera.sensors.icarus2 Namespace Reference

Classes

- class [icarus2](#)

5.12.1 Detailed Description

Parameters and functions specific to the four-frame icarus2 sensor

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

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

5.13 nsCamera.sensors.sensorBase Namespace Reference

Classes

- class [sensorBase](#)

5.13.1 Detailed Description

Superclass for nsCamera sensors

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

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

5.14 nsCamera.utils Namespace Reference

Namespaces

- namespace [crc16pure](#)
- namespace [FlatField](#)
- namespace [misc](#)
- namespace [Packet](#)
- namespace [Subregister](#)

Variables

- list `__all__` = ["SubRegister", "Packet", "FlatField", "misc"]

5.14.1 Detailed Description

This package is a collection of utility classes for the CameraAssembler

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

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

5.14.2 Variable Documentation

5.14.2.1 `__all__`

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

Definition at line 30 of file `__init__.py`.

5.15 nsCamera.utils.crc16pure Namespace Reference

Functions

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

Variables

- list `CRC16_XMODEM_TABLE`

5.15.1 Detailed Description

Pure python library for calculating CRC16

NOTE: modified slightly to combine Python 2 and Python 3 versions in single file

5.15.2 Function Documentation

5.15.2.1 `_crc16()`

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

Calculate CRC16 using the given table.

`data` - data for calculating CRC, must be a string

`crc` - initial value

`table` - table for calculating CRC (list of 256 integers)

Return calculated value of CRC

Definition at line 299 of file [crc16pure.py](#).

```
00299 def _crc16(data, crc, table):
00300     """Calculate CRC16 using the given table.
00301     `data` - data for calculating CRC, must be a string
00302     `crc` - initial value
00303     `table` - table for calculating CRC (list of 256 integers)
00304     Return calculated value of CRC
00305     """
00306     for byte in data:
00307         if sys.version_info > (3,):
00308             crc = ((crc << 8) & 0xFF00) ^ table[((crc >> 8) & 0xFF) ^ byte]
00309         else:
00310             crc = ((crc << 8) & 0xFF00) ^ table[((crc >> 8) & 0xFF) ^ ord(byte)]
00311
00312     return crc & 0xFFFF
00313
00314
```

5.15.2.2 `crc16xmodem()`

```
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](#).

```
00315 def crc16xmodem(data, crc=0):
00316     """Calculate CRC-CCITT (XModem) variant of CRC16.
00317     `data` - data for calculating CRC, must be a string
00318     `crc` - initial value
00319     Return calculated value of CRC
00320     """
00321     return _crc16(data, crc, CRC16_XMODEM_TABLE)
```

5.15.3 Variable Documentation

5.15.3.1 `CRC16_XMODEM_TABLE`

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

Definition at line 39 of file [crc16pure.py](#).

5.16 nsCamera.utils.FlatField Namespace Reference

Functions

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

5.16.1 Detailed Description

Functions for batch flat-field image corrections

Do not use this file as a template for new code development

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

Author: Matthew Dayton (dayton5@llnl.gov)

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

5.16.2 Function Documentation

5.16.2.1 generateFF()

```
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](#).

```
00063 ):
00064     # TODO: documentation
00065     # use of ROI here not compatible with use of ROI in removeFF
00066
00067     if directory:
00068         cwd = os.getcwd()
00069         newpath = os.path.join(cwd, directory)
00070         os.chdir(newpath)
00071     if not FRAMES:
00072         print("No framelist provided, defaulting to four frames")
00073         FRAMES = ["Frame_0", "Frame_1", "Frame_2", "Frame_3"]
00074     for f in FRAMES:
00075         files = getFilenames(frame=f)
00076         imgslist = [getROIvector(fn, roi) for fn in files] # a list of flattened images
00077         imgsarray = np.vstack(imgslist) # turn the list into an array
00078         npix = np.shape(imgsarray)[1] # total number of pixels
00079         x = np.median(imgsarray, axis=1) # median of each image used for flat fielding
00080         y = []
00081         for i in range(npix):
00082             # each member of y represents a pixel, as a list of magnitudes over all the
00083             # images
00084             y.append(imgsarray[:, i])
00085         # get pixel gain and offset for flatfield ff using Thiel-Sen slopes
00086         ff = []
00087         ff = parallel.Parallel(n_jobs=ncores, verbose=5, pre_dispatch="2 * n_jobs")(
00088             delayed(tslopes)(x, pixel) for pixel in y
00089         )
00090         # x is the dependent variable; here uses median of image as characteristic of
00091         # noise level
00092         m, c = zip(*ff) # separate into gain and offset
00093         m = np.array(m)
00094         m[m < 0.1] = 0.1 # handle outliers
00095         m[m > 1000] = 1000 # handle outliers
00096         m = 1.0 / m
00097         m = m.reshape(roi[3] - roi[1], roi[2] - roi[0]) # turn into matrix
00098         c = np.array(c).reshape(roi[3] - roi[1], roi[2] - roi[0]) # turn into matrix
00099
00100         with open("px_gain%s.txt" % f.replace("Frame_", "f"), "w+") as file:
00101             np.savetxt(file, m)
00102         with open("px_off%s.txt" % f.replace("Frame_", "f"), "w+") as file:
00103             np.savetxt(file, c)
00104
00105
```

5.16.2.2 getFilenames()

```
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](#).

```
00032 def getFilenames(frame="Frame 1"):
00033     """
00034     get a list of tiff filenames in current working director for frame
00035     """
00036     onlyfiles = next(os.walk("./"))[2]
00037     return [k for k in onlyfiles if frame in k and "tif" in k]
00038
00039
```

5.16.2.3 getROIvector()

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

return a numpy row vector of version of the image

Definition at line 40 of file [FlatField.py](#).

```
00040 def getROIvector(imgfilename, roi):
00041     """
00042     return a numpy row vector of version of the image
00043     """
00044     img = imread(imgfilename)
00045     vroi = img[(roi[1]) : (roi[3]), (roi[0]) : (roi[2])].flattenlist()
00046     return vroi
00047
00048
```

5.16.2.4 removeFF()

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

Definition at line 106 of file [FlatField.py](#).

```
00106 def removeFF(filename, directory="", roi=[0, 0, 512, 1024]):
00107     if directory:
00108         cwd = os.getcwd()
00109         newpath = os.path.join(cwd, directory)
00110         os.chdir(newpath)
00111         framenum = re.search("Frame_(\\d)", filename).group(1)
00112         gainFilename = "px_gain_f" + framenum + ".txt"
00113         gainall = np.loadtxt(gainFilename)
00114         gain = gainall[(roi[1]) : (roi[3]), (roi[0]) : (roi[2])]
00115         offFilename = "px_off_f" + framenum + ".txt"
00116         offsetall = np.loadtxt(offFilename, dtype="uint32")
00117         offset = offsetall[(roi[1]) : (roi[3]), (roi[0]) : (roi[2])]
00118
00119         beforeImageall = imread(filename)
00120         beforeImage = beforeImageall[(roi[1]) : (roi[3]), (roi[0]) : (roi[2])]
00121         imageMed = np.median(beforeImage)
00122
00123         flat = imageMed * gain + offset
00124         flat = flat.clip(0)
00125         fix = beforeImage - flat
00126         clipped = fix.clip(0)
00127         fixinit = clipped.astype("uint16")
00128         fiximg = Image.fromarray(fixinit)
00129
00130         fixFilename = filename[:-4] + ".ff" + filename[-4:]
00131         fiximg.save(fixFilename)
00132
```

5.16.2.5 removeFFall()

```
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](#).

```
00137 ):
00138     cwd = os.getcwd()
00139     if directory:
00140         newpath = os.path.join(cwd, directory)
00141     else:
00142         newpath = cwd
00143     os.chdir(newpath)
00144     files = next(os.walk("./"))[2]
00145     filelist = []
00146     for frame in FRAMES:
00147         filelist.extend([k for k in files if frame in k and "tif" in k])
00148     for fname in filelist:
00149         removeFF(fname, directory, roi)
00150
00151
```

5.16.2.6 tslopes()

```
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](#).

```
00049 def tslopes(x, y):
00050     """
00051     theilslopes implements a method for robust linear regression.
00052     It computes the slope as the median of all slopes between paired values.
00053     """
00054     val = theilslopes(x, y)
00055     return [val[0], val[1]]
00056
00057
```

5.16.3 Variable Documentation

5.16.3.1 action

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

Definition at line 157 of file [FlatField.py](#).

5.16.3.2 args

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

Definition at line 167 of file [FlatField.py](#).

5.16.3.3 default

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

Definition at line 157 of file [FlatField.py](#).

5.16.3.4 dest

`nsCamera.utils.FlatField.dest`

Definition at line 157 of file [FlatField.py](#).

5.16.3.5 directory

`nsCamera.utils.FlatField.directory`

Definition at line 169 of file [FlatField.py](#).

5.16.3.6 framelist

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

Definition at line 168 of file [FlatField.py](#).

5.16.3.7 help

`nsCamera.utils.FlatField.help`

Definition at line 157 of file [FlatField.py](#).

5.16.3.8 nargs

`nsCamera.utils.FlatField.nargs`

Definition at line 161 of file [FlatField.py](#).

5.16.3.9 parser

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

Definition at line 155 of file [FlatField.py](#).

5.17 nsCamera.utils.misc Namespace Reference

Classes

- class [fakeCA](#)

Functions

- [makeLogLabels](#) (logtag, label)
- [getEnter](#) (text)
- [checkCRC](#) (rval)
- [str2bytes](#) (astring)
- [bytes2str](#) (bytesequences)
- [str2nparray](#) (valstring)
- [flattenlist](#) (x)
- [generateFrames](#) (camassem, data, columns=1)
- [loadDumpedData](#) (filename="frames.txt", path=None, filetype="txt", sensor="daedalus", firstframe=None, lastframe=None, width=None, height=None, padToFull=None, firstrow=None, lastrow=None, maxwidth=None, maxheight=None, bytesperpixel=None, interlacing=None, columns=1)
- [saveTiffs](#) (self, frames, path=None, filename="Frame", prefix=None, index=None)
- [plotFrames](#) (self, frames, index=None)
- [partition](#) (self, frames, columns)

5.17.1 Detailed Description

Miscellaneous utilities, including batch processing of images acquired using the `nsCamera`. These are functions that don't require a `cameraAssembler` object to be instantiated before use.

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

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

5.17.2 Function Documentation

5.17.2.1 `bytes2str()`

```
nsCamera.utils.misc.bytes2str (
    bytesequences )
```

Python-version-agnostic converter of bytes to hexadecimal strings

Args:

bytesequences: sequence of bytes as string (Py2) or bytes (Py3)

Returns:

hexadecimal string representation of 'bytes' without '0x'

Definition at line 123 of file [misc.py](#).

```
00123 def bytes2str(bytessequence):
00124     """
00125     Python-version-agnostic converter of bytes to hexadecimal strings
00126
00127     Args:
00128         bytessequence: sequence of bytes as string (Py2) or bytes (Py3)
00129
00130     Returns:
00131         hexadecimal string representation of 'bytes' without '0x'
00132     """
00133     try:
00134         estring = binascii.b2a_hex(bytessequence)
00135     except TypeError:
00136         logging.error(
00137             "ERROR: bytes2str: Invalid byte sequence: '{bytessequence}'; returning an"
00138             " empty string".format(bytessequence=bytessequence)
00139         )
00140     return ""
00141 python, _, _, _, _ = sys.version_info
00142 if python >= 3:
00143     estring = str(estring)[2:-1]
00144 return estring
00145
00146
```

5.17.2.2 checkCRC()

```
nsCamera.utils.misc.checkCRC (
    rval )
```

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

Args:
rval: hexadecimal string

Returns:
boolean, True if CRCs match, False if they don't match or the input is invalid

Definition at line 72 of file [misc.py](#).

```
00072 def checkCRC(rval):
00073     """
00074     Calculate CRC for rval[:-4] and compare with expected CRC in rval[-4:]
00075
00076     Args:
00077         rval: hexadecimal string
00078
00079     Returns:
00080         boolean, True if CRCs match, False if they don't match or the input is invalid
00081     """
00082     if not isinstance(rval, str) or len(rval) < 5:
00083         logging.error("ERROR: checkCRC: Invalid input: {rval}".format(rval=rval))
00084     return False
00085     data_crc = int(rval[-4:], base=16)
00086     CRC_calc = crc16pure.crc16xmodem(str2bytes(rval[:-4]))
00087     return CRC_calc == data_crc
00088
00089
```

5.17.2.3 flattenlist()

```
nsCamera.utils.misc.flattenlist (
    x )
```

Flatten list of lists recursively into single list

Definition at line 179 of file [misc.py](#).

```

00179 def flattenlist(x):
00180     """
00181     Flatten list of lists recursively into single list
00182     """
00183     python, _, _, _, _ = sys.version_info
00184     try:
00185         if python >= 3:
00186             if isinstance(x, collections.abc.Iterable):
00187                 return [a for i in x for a in flattenlist(i)]
00188             else:
00189                 return [x]
00190         else:
00191             if isinstance(x, collections.Iterable):
00192                 return [a for i in x for a in flattenlist(i)]
00193             else:
00194                 return [x]
00195     except RecursionError:
00196         logging.error(
00197             "ERROR: flattenlist: input '{x}' is pathological and cannot be flattened."
00198             " Attempting to return the input unchanged"
00199         )
00200     return x
00201
00202

```

5.17.2.4 generateFrames()

```

nsCamera.utils.misc.generateFrames (
    camassem,
    data,
    columns = 1 )

```

Processes data stream from board into frames and applies sensor-specific parsing.
 Generates padded data for full-size option of setRows.
 If used for offline processing, replace the 'self' object with the parameter object returned by loadDumpedData().
 If the data stream is incomplete (e.g., from an interrupted download), the data is padded with zeros to the correct length.

Args:

self: during normal operation, is the CameraAssembler object. During offline operation, is a parameters object as returned by loadDumpedData()
 data: text stream from board or loaded file, or numpy from loaded file
 columns: 1 for single image, 2 for separate hemisphere images

Returns: list of parsed frames

Definition at line 261 of file [misc.py](#).

```

00261 def generateFrames(camassem, data, columns=1):
00262     """
00263     Processes data stream from board into frames and applies sensor-specific parsing.
00264     Generates padded data for full-size option of setRows.
00265     If used for offline processing, replace the 'self' object with the parameter object
00266     returned by loadDumpedData().
00267     If the data stream is incomplete (e.g., from an interrupted download), the data is
00268     padded with zeros to the correct length.
00269
00270     Args:
00271         self: during normal operation, is the CameraAssembler object. During offline
00272             operation, is a parameters object as returned by loadDumpedData()
00273         data: text stream from board or loaded file, or numpy from loaded file
00274         columns: 1 for single image, 2 for separate hemisphere images
00275
00276     Returns: list of parsed frames
00277     """
00278     logging.debug("DEBUG" + camassem.logtag + ": generateFrames")
00279     if isinstance(data[0], str):

```

```

00280         allframes = str2nparray(data)
00281     else:
00282         allframes = data
00283     nframes = camassem.sensor.lastframe - camassem.sensor.firstframe + 1
00284     frames = [0] * nframes
00285     framesize = camassem.sensor.width * (
00286         camassem.sensor.lastrow - camassem.sensor.firstrow + 1
00287     )
00288     if hasattr(camassem, "ca"):
00289         padIt = camassem.ca.padToFull
00290     else:
00291         padIt = camassem.padToFull
00292     if padIt:
00293         toprows = camassem.sensor.firstrow
00294         botrows = (camassem.sensor.maxheight - 1) - camassem.sensor.lastrow
00295         padtop = np.zeros(toprows * camassem.sensor.maxwidth, dtype=int)
00296         padbot = np.zeros(botrows * camassem.sensor.maxwidth, dtype=int)
00297         for n in range(nframes):
00298             thisframe = np.concatenate(
00299                 (padtop, allframes[n * framesize : (n + 1) * framesize]), padbot)
00300             frames[n] = thisframe
00301     else:
00302         for n in range(nframes):
00303             frames[n] = allframes[n * framesize : (n + 1) * framesize]
00304     # self.clearStatus()
00305     parsed = camassem.sensor.parseReadoff(frames, columns)
00306     return parsed
00307
00308
00309

```

5.17.2.5 getEnter()

```

nsCamera.utils.misc.getEnter (
    text )

```

Wait for enter key to be pressed.

Args:
text: message asking for keypress

Definition at line 58 of file [misc.py](#).

```

00058 def getEnter(text):
00059     """
00060     Wait for enter key to be pressed.
00061
00062     Args:
00063         text: message asking for keypress
00064     """
00065     python, _, _, _ = sys.version_info
00066     if python >= 3:
00067         input(text)
00068     else:
00069         raw_input(text)
00070
00071

```

5.17.2.6 loadDumpedData()

```

nsCamera.utils.misc.loadDumpedData (
    filename = "frames.txt",
    path = None,
    filetype = "txt",
    sensor = "daedalus",

```

```

    firstframe = None,
    lastframe = None,
    width = None,
    height = None,
    padToFull = None,
    firstrow = None,
    lastrow = None,
    maxwidth = None,
    maxheight = None,
    bytesperpixel = None,
    interlacing = None,
    columns = 1 )

```

`_summary_`

```

Output can be piped directly to saveTiffs:
    pars,frames=loadDumpedData(filename="Dump.npy")
    saveTiffs(pars,frames)

```

Args:

```

filename (str, optional): _description_. Defaults to "frames.txt".
path (_type_, optional): _description_. Defaults to None.
filetype (str, optional): _description_. Defaults to "txt".
sensor (str, optional): _description_. Defaults to "daedalus".
firstframe (_type_, optional): _description_. Defaults to None.
lastframe (_type_, optional): _description_. Defaults to None.
width (_type_, optional): _description_. Defaults to None.
height (_type_, optional): _description_. Defaults to None.
padToFull (_type_, optional): _description_. Defaults to None.
firstrow (_type_, optional): _description_. Defaults to None.
lastrow (_type_, optional): _description_. Defaults to None.
maxwidth (_type_, optional): _description_. Defaults to None.
maxheight (_type_, optional): _description_. Defaults to None.
interlacing (_type_, optional): _description_. Defaults to None.
columns (int, optional): _description_. Defaults to 1.

```

Returns:

```

Tuple (parameter object, list of data frames)

```

Definition at line 310 of file `misc.py`.

```

00327 ):
00328     """_summary_
00329
00330     Output can be piped directly to saveTiffs:
00331         pars,frames=loadDumpedData(filename="Dump.npy")
00332         saveTiffs(pars,frames)
00333     Args:
00334         filename (str, optional): _description_. Defaults to "frames.txt".
00335         path (_type_, optional): _description_. Defaults to None.
00336         filetype (str, optional): _description_. Defaults to "txt".
00337         sensor (str, optional): _description_. Defaults to "daedalus".
00338         firstframe (_type_, optional): _description_. Defaults to None.
00339         lastframe (_type_, optional): _description_. Defaults to None.
00340         width (_type_, optional): _description_. Defaults to None.
00341         height (_type_, optional): _description_. Defaults to None.
00342         padToFull (_type_, optional): _description_. Defaults to None.
00343         firstrow (_type_, optional): _description_. Defaults to None.
00344         lastrow (_type_, optional): _description_. Defaults to None.
00345         maxwidth (_type_, optional): _description_. Defaults to None.
00346         maxheight (_type_, optional): _description_. Defaults to None.
00347         interlacing (_type_, optional): _description_. Defaults to None.
00348         columns (int, optional): _description_. Defaults to 1.
00349
00350
00351     Returns:
00352         Tuple (parameter object, list of data frames)
00353     """

```

```

00354     logging.debug("DEBUG: loadDumpedData")
00355     if sensor.lower() == "daedalus":
00356         import nsCamera.sensors.daedalus as snsr
00357     elif sensor.lower() == "icarus":
00358         import nsCamera.sensors.icarus as snsr
00359     elif sensor.lower() == "icarus2":
00360         import nsCamera.sensors.icarus2 as snsr
00361     else:
00362         logging.error(
00363             "ERROR loadDumpedData: invalid sensor type provided, defaulting to icarus2"
00364         )
00365         import nsCamera.sensors.icarus2 as snsr
00366
00367     def buildEmptyFrames():
00368         cols = [0] * 512
00369         frame = np.array([cols] * (lastrow - firstrow + 1))
00370         return [frame] * (lastframe - firstframe + 1)
00371
00372     # get defaults from class declarations if not specified as parameter
00373     if firstframe is None:
00374         firstframe = snsr.firstframe
00375     if lastframe is None:
00376         lastframe = snsr.lastframe
00377     # TODO: first frame number
00378     if width is None:
00379         width = snsr.width
00380     if height is None:
00381         height = snsr.height
00382     if firstrow is None:
00383         firstrow = snsr.firstrow
00384     if lastrow is None:
00385         lastrow = snsr.lastrow
00386     if maxwidth is None:
00387         maxwidth = snsr.maxwidth
00388     if maxheight is None:
00389         maxheight = snsr.maxheight
00390     if bytesperpixel is None:
00391         bytesperpixel = snsr.bytesperpixel
00392     if interlacing is None:
00393         interlacing = snsr.interlacing
00394
00395     parameters = fakeCA(
00396         sensor,
00397         firstframe,
00398         lastframe,
00399         width,
00400         height,
00401         padToFull,
00402         firstrow,
00403         lastrow,
00404         maxwidth,
00405         maxheight,
00406         bytesperpixel,
00407         interlacing,
00408     )
00409
00410     if path is None:
00411         path = os.path.join(os.getcwd())
00412     datafile = os.path.join(path, filename)
00413     if filename[-3:].lower() == "txt":
00414         filetype = "txt"
00415     elif filename[-3:].lower() == "npz":
00416         filetype = "npz"
00417     # TODO: return empty frames if error
00418
00419     if filetype == "npz":
00420         expectedlength = (lastframe - firstframe + 1) * (lastrow - firstrow + 1) * width
00421         try:
00422             f = np.load(datafile)
00423             padding = expectedlength - len(f)
00424             if padding:
00425                 logging.warning(
00426                     "{logwarn}loadDumpedData: Payload is shorter than expected."
00427                     " Padding with '0's".format(logwarn=parameters.logwarn)
00428                 )
00429             f = np.pad(f, (0, padding), "constant", constant_values=(0))
00430             frames = generateFrames(parameters, f, columns)
00431             return parameters, frames
00432
00433         except OSError as err:
00434             logging.error(

```

```

00435         "{logerr}loadDumpedData: OS error: {err}. Returning empty"
00436         " frames.".format(logerr=parameters.logerr, err=err)
00437     )
00438     return parameters, buildEmptyFrames()
00439 except:
00440     logging.error(
00441         "{logerr}loadDumpedData: Unexpected error: {err}. Returning empty"
00442         " frames.".format(logerr=parameters.logerr, err=str(sys.exc_info()[0]))
00443     )
00444     return parameters, buildEmptyFrames()
00445 # if filetype is not explicitly npy, try loading as text
00446 else:
00447     # Payload size as string implied by provided parameters
00448     expectedlength = (
00449         4 * (lastframe - firstframe + 1) * (lastrow - firstrow + 1) * width
00450     )
00451
00452     try:
00453         f = open(datafile, "r")
00454         s = f.read()
00455
00456         padding = expectedlength - len(s)
00457         if padding:
00458             logging.warning(
00459                 "{logwarn}loadDumpedData: Payload is shorter than expected."
00460                 " Padding with '0's".format(logwarn=parameters.logwarn)
00461             )
00462             s = s.ljust(expectedlength, "0")
00463
00464         frames = generateFrames(parameters, s)
00465         return parameters, frames
00466
00467 except OSError as err:
00468     logging.error(
00469         "{logerr}loadDumpedData: OS error: {err}. Returning empty"
00470         " frames.".format(logerr=parameters.logerr, err=err)
00471     )
00472     return parameters, buildEmptyFrames()
00473 except ValueError:
00474     logging.error(
00475         "{logerr}loadDumpedData: Could not convert data to an integer."
00476         " Returning empty frames.".format(logerr=parameters.logerr)
00477     )
00478     return parameters, buildEmptyFrames()
00479 except:
00480     logging.error(
00481         "{logerr}loadDumpedData: Unexpected error: {err}. Returning empty"
00482         " frames.".format(logerr=parameters.logerr, err=str(sys.exc_info()[0]))
00483     )
00484     return parameters, buildEmptyFrames()
00485
00486

```

5.17.2.7 makeLogLabels()

```

nsCamera.utils.misc.makeLogLabels (
    logtag,
    label )

```

Definition at line 39 of file `misc.py`.

```

00039 def makeLogLabels(logtag, label):
00040     if logtag is None:
00041         logtag = ""
00042
00043     logcritbase = "CRITICAL{logtag}:".format(logtag=logtag)
00044     logerrbase = "ERROR{logtag}:".format(logtag=logtag)
00045     logwarnbase = "WARNING{logtag}:".format(logtag=logtag)
00046     loginfobase = "INFO{logtag}:".format(logtag=logtag)
00047     logdebugbase = "DEBUG{logtag}:".format(logtag=logtag)
00048
00049     logcrit = "{base}{label}".format(base=logcritbase, label=label)
00050     logerr = "{base}{label}".format(base=logerrbase, label=label)
00051     logwarn = "{base}{label}".format(base=logwarnbase, label=label)
00052     loginfo = "{base}{label}".format(base=loginfobase, label=label)

```

```

00053     logdebug = "{base}{label}".format(base=logdebugbase, label=label)
00054
00055     return logcrit, logerr, logwarn, loginfo, logdebug
00056
00057

```

5.17.2.8 partition()

```

nsCamera.utils.misc.partition (
    self,
    frames,
    columns )

```

Extracts interlaced frames and divides images by hemispheres. If interlacing does not evenly divide the height, remainder lines will be dropped

Args:

```

self: during normal operation, is sensor object. During offline
      operation, is the parameter.sensor object returned by loadDumpedData()
frames: list of full-sized frames
columns: 1 for single image, 2 for separate hemisphere images

```

Returns: list of deinterlaced frames

Definition at line 634 of file [misc.py](#).

```

00634 def partition(self, frames, columns):
00635     """
00636     Extracts interlaced frames and divides images by hemispheres. If interlacing does
00637     not evenly divide the height, remainder lines will be dropped
00638
00639     Args:
00640         self: during normal operation, is sensor object. During offline
00641               operation, is the parameter.sensor object returned by loadDumpedData()
00642         frames: list of full-sized frames
00643         columns: 1 for single image, 2 for separate hemisphere images
00644
00645     Returns: list of deinterlaced frames
00646     """
00647     logging.debug(
00648         "{logdebug}partition: columns = {columns}, interlacing = {interlacing}".format(
00649             logdebug=self.logdebug, columns=columns, interlacing=self.sensor.interlacing
00650         )
00651     )
00652
00653     def unshuffle(frames, ifactor):
00654         warntrimmed = False
00655         if self.padToFull:
00656             newheight = self.sensor.maxheight // (ifactor + 1)
00657             if newheight != (self.sensor.maxheight / (ifactor + 1)):
00658                 warntrimmed = True
00659         else:
00660             newheight = self.sensor.height // (ifactor + 1)
00661             if newheight != (self.sensor.height / (ifactor + 1)):
00662                 warntrimmed = True
00663
00664         if warntrimmed:
00665             logging.warning(
00666                 "{logwarn} partition: interlacing setting requires dropping of lines to"
00667                 " maintain consistent frame sizes ".format(logwarn=self.logwarn)
00668             )
00669         delaced = []
00670         for frame in frames:
00671             for sub in range(ifactor + 1):
00672                 current = np.zeros((newheight, self.sensor.width // columns), dtype=int)
00673                 for line in range(newheight):
00674                     current[line] = frame[(ifactor + 1) * line + sub]
00675                 delaced.append(current)
00676         nframes = self.sensor.lastframe - self.sensor.firstframe + 1
00677         resorted = [None] * len(delaced)

```

```

00678         for sub in range(ifactor + 1):
00679             for idx, frame in enumerate(frames):
00680                 resorted[sub * nframes + idx] = delaced[idx * (ifactor + 1) + sub]
00681         return resorted
00682
00683     if self.sensor.interlacing[0] != self.sensor.interlacing[1]:
00684         columns = 2 # true even if not explicitly requested by readoff
00685     if columns == 1:
00686         if self.sensor.interlacing == [0, 0]: # don't do anything
00687             return frames
00688         else:
00689             return unshuffle(frames, self.sensor.interlacing[0])
00690     else:
00691         # reshape frame into the proper shape, then split horizontally
00692         if self.padToFull:
00693             framesab = [
00694                 np.hsplit(frame.reshape(self.sensor.maxheight, -1), 2)
00695                 for frame in frames
00696             ]
00697         else:
00698             framesab = [
00699                 np.hsplit(
00700                     frame.reshape((self.sensor.lastrow - self.sensor.firstrow + 1), -1),
00701                     2,
00702                 )
00703                 for frame in frames
00704             ]
00705         framesa = [hemis[0] for hemis in framesab]
00706         framesb = [hemis[1] for hemis in framesab]
00707     if self.sensor.interlacing == [0, 0]:
00708         return framesa + framesb
00709     else:
00710         return unshuffle(framesa, self.sensor.interlacing[0]) + unshuffle(
00711             framesb, self.sensor.interlacing[1]
00712         )
00713
00714 """
00715 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00716 LLNL-CODE-838080
00717
00718 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00719 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00720 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00721 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00722 be made under this license.
00723 """
00724 """

```

5.17.2.9 plotFrames()

```

nsCamera.utils.misc.plotFrames (
    self,
    frames,
    index = None )

```

Plot frame or list of frames as individual graphs.

Args:

self: during normal operation, is cameraAssembler object. During offline operation, is the parameter object returned by loadDumpedData()
frames: numpy array or list of numpy arrays
index: number to start frame numbering

Returns:

Error string

Definition at line 567 of file [misc.py](#).

```

00567 def plotFrames(self, frames, index=None):
00568     """

```



```

00569     Plot frame or list of frames as individual graphs.
00570
00571     Args:
00572         self: during normal operation, is cameraAssembler object. During offline
00573             operation, is the parameter object returned by loadDumpedData()
00574         frames: numpy array or list of numpy arrays
00575         index: number to start frame numbering
00576
00577     Returns:
00578         Error string
00579     """
00580     # logging.info(self.loginfo + "plotFrames: index = " + str(index))
00581     logging.info(
00582         "{loginfo}: plotFrames: index = {index}".format(
00583             loginfo=self.loginfo, index=index
00584         )
00585     )
00586     err = ""
00587     if index is None:
00588         nframe = self.sensor.firstframe
00589     else:
00590         nframe = index
00591
00592     if not isinstance(frames, list):
00593         frames = [frames]
00594
00595     # if this is a text string from fast readoff, do the numpy conversion now
00596     if isinstance(frames[0], str):
00597         frames = generateFrames(frames)
00598
00599     framestemp = np.copy(frames)
00600     for idx, frame in enumerate(framestemp):
00601         if idx < len(framestemp) / 2:
00602             interlacing = self.sensor.interlacing[0]
00603         else:
00604             interlacing = self.sensor.interlacing[1]
00605         try:
00606             if self.padToFull:
00607                 frame = np.reshape(
00608                     frame, (self.sensor.maxheight // (interlacing + 1), -1)
00609                 )
00610             else:
00611                 frame = np.reshape(
00612                     frame,
00613                     (
00614                         (self.sensor.lastrow - self.sensor.firstrow + 1)
00615                         // (interlacing + 1),
00616                         -1,
00617                     ),
00618                 )
00619         except:
00620             err = "{logerr}plotFrames: unable to plot frame".format(logerr=self.logerr)
00621             logging.error(err)
00622             continue
00623         plt.imshow(frame, cmap="gray")
00624         name = "Frame %d" % nframe
00625         plt.title(name)
00626         plt.show()
00627         nframe += 1
00628     return err
00629
00630
00631 # TODO: separate images for hemispheres with different timing
00632
00633

```

5.17.2.10 saveTiffs()

```

nsCamera.utils.misc.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. If used for standalone, use the parameter object returned by loadDumpedData() as the first argument

Args:

self: during normal operation, is cameraAssembler object. During offline operation, is the parameter object returned by loadDumpedData()
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 487 of file misc.py.

```

00494 ):
00495     """
00496     Save numpy array or list of numpy arrays or single array to disk as individual
00497     tiffs, with frame number appended to filename. If used for standalone, use the
00498     parameter object returned by loadDumpedData() as the first argument
00499
00500     Args:
00501         self: during normal operation, is cameraAssembler object. During offline
00502             operation, is the parameter object returned by loadDumpedData()
00503         frames: numpy array or list of numpy arrays
00504         path: save path, defaults to './output'
00505         filename: defaults to 'Frame' followed by frame number
00506         prefix: prepended to 'filename', defaults to time/date
00507             (e.g. '160830-124704_')
00508         index: number to start frame numbering
00509
00510     Returns:
00511         Error string
00512     """
00513     # logging.info("INFO" + self.logtag + ": saveTiffs")
00514     logging.info("{loginfo}: saveTiffs".format(loginfo=self.logininfo))
00515     err = ""
00516     if path is None:
00517         path = os.path.join(os.getcwd(), "output")
00518     if prefix is None:
00519         prefix = datetime.now().strftime("%Y%m%d-%H%M%S%f")[:-5] + "_"
00520     if not os.path.exists(path):
00521         os.makedirs(path)
00522     if index is None:
00523         firstnum = self.sensor.firstframe
00524     else:
00525         firstnum = index
00526
00527     # if this is a text string from fast readoff, do the numpy conversion now
00528     if isinstance(frames[0], str):
00529         frames = generateFrames(frames)
00530
00531     framestemp = np.copy(frames)
00532     if np.issubdtype(type(framestemp[0]), np.number):
00533         # if type(framestemp[0]) is np.uint16:
00534         # single frame needs to be a list containing one frame
00535         framestemp = [framestemp]
00536     for idx, frame in enumerate(framestemp):
00537         if idx < len(framestemp) / 2:
00538             interlacing = self.sensor.interlacing[0]
00539         else:
00540             interlacing = self.sensor.interlacing[1]
00541     try:
00542         if self.padToFull:
00543             frame = np.reshape(
00544                 frame, (self.sensor.maxheight // (interlacing + 1), -1)
00545             )
00546         else:
00547             frame = np.reshape(
00548                 frame,
00549                 (
00550                     (self.sensor.lastrow - self.sensor.firstrow + 1)
00551                     // (interlacing + 1),

```

```

00552             -1,
00553         ),
00554     )
00555     frameimg = Image.fromarray(frame, "I;16")
00556     namenun = filename + "_%d" % firstnum
00557     tifpath = os.path.join(path, prefix + namenun + ".tif")
00558     frameimg.save(tifpath)
00559     firstnum += 1
00560 except Exception:
00561     err = "saveTiffs: unable to save images"
00562     # logging.error("ERROR" + self.logtag + ": " + err)
00563     logging.error("{logerr}: {err}".format(logerr=self.logerr, err=err))
00564 return err
00565
00566

```

5.17.2.11 str2bytes()

```

nsCamera.utils.misc.str2bytes (
    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 90 of file [misc.py](#).

```

00090 def str2bytes(astring):
00091     """
00092     Python-version-agnostic converter of hexadecimal strings to bytes
00093
00094     Args:
00095         astring: hexadecimal string without '0x'
00096
00097     Returns:
00098         byte string equivalent to input string
00099     """
00100
00101     python, _, _, _, _ = sys.version_info
00102     if python >= 3:
00103         try:
00104             dbytes = binascii.a2b_hex(astring)
00105         except:
00106             logging.error(
00107                 "ERROR: str2bytes: invalid input: '{astring}'; returning zero"
00108                 " byte".format(astring=astring)
00109             )
00110             dbytes = b"\x00"
00111     else:
00112         try:
00113             dbytes = astring.decode("hex")
00114         except:
00115             logging.error(
00116                 "ERROR: str2bytes: invalid input: '{astring}'; returning zero "
00117                 " byte".format(astring=astring)
00118             )
00119             dbytes = b"\x00"
00120     return dbytes
00121
00122

```

5.17.2.12 str2nparray()

```
nsCamera.utils.misc.str2nparray (
    valstring )
```

Convert string into array of uint16s

Args:

valstring: string of hexadecimal characters

Returns:

numpy array of uint16

Definition at line 147 of file [misc.py](#).

```
00147 def str2nparray(valstring):
00148     """
00149     Convert string into array of uint16s
00150
00151     Args:
00152         valstring: string of hexadecimal characters
00153
00154     Returns:
00155         numpy array of uint16
00156     """
00157     if not isinstance(valstring, str):
00158         logging.error(
00159             "ERROR: str2nparray: Invalid input: {valstring} is not a string. Returning"
00160             " an empty array".format(valstring=valstring)
00161         )
00162         return np.array([])
00163     stringlen = len(valstring)
00164     arraylen = int(stringlen / 4)
00165     outarray = np.empty(int(arraylen), dtype="uint16")
00166
00167     for i in range(0, arraylen):
00168         try:
00169             outarray[i] = int(valstring[4 * i : 4 * i + 4], 16)
00170         except ValueError:
00171             logging.error(
00172                 "ERROR: str2nparray: input string does not represent a hexadecimal"
00173                 " integer. Returning an empty array"
00174             )
00175         return np.array([])
00176     return outarray
00177
00178
```

5.18 nsCamera.utils.Packet Namespace Reference

Classes

- class [Packet](#)

5.18.1 Detailed Description

Packet object for communication with boards

Author: Brad Funsten (funsten1@llnl.gov)

Author: Jeremy Hill (hill35@llnl.gov)

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

5.19 nsCamera.utils.Subregister Namespace Reference

Classes

- class [SubRegister](#)

5.19.1 Detailed Description

Subregister object represents a subset of a full register

Author: Matthew Dayton (dayton5@llnl.gov)

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

Copyright (c) 2025, 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. 'nsCamera' is distributed under the terms of the MIT license. All new contributions must be made under this license.

Version: 2.1.2 (February 2025)

Chapter 6

Class Documentation

6.1 nsCamera.CameraAssembler.CameraAssembler Class Reference

Public Member Functions

- `__init__` (self, `boardname`="lInl_v4", `commname`="GigE", `sensorname`="icarus2", `verbose`=4, `port`=None, `ip`=None, `logfile`=None, `logtag`=None, `timeout`=30)
- `initBoard` (self)
Aliases to other objects' methods TODO: properly delegate these methods.
- `initPots` (self)
- `latchPots` (self)
- `initSensor` (self)
- `configADCs` (self)
- `disarm` (self)
- `startCapture` (self, mode)
- `readSRAM` (self)
- `waitForSRAM` (self, `timeout`=None)
- `getTimer` (self)
- `resetTimer` (self)
- `enableLED` (self, status=1)
- `setLED` (self, LED=1, status=1)
- `setPowerSave` (self, status=1)
- `setPPER` (self, pollperiod=None)
- `getTemp` (self, scale=None)
- `getPressure` (self, offset=None, sensitivity=None, units=None)
- `clearStatus` (self)
- `checkStatus` (self)
- `checkStatus2` (self)
- `reportStatus` (self)
- `reportEdgeDetects` (self)
- `dumpStatus` (self)
- `checkSensorVoltStat` (self)
- `setTiming` (self, side="AB", sequence=None, delay=0)
- `setArbTiming` (self, side="AB", sequence=None)

- [getTiming](#) (self, side=None, actual=None)
- [setManualShutters](#) (self, timing=None)
- [setManualTiming](#) (self, timing=None)
- [getManualTiming](#) (self)
- [getSensTemp](#) (self, scale=None, offset=None, slope=None, dec=1)
- [sensorSpecific](#) (self)
- [selectOscillator](#) (self, osc=None)
- [setInterlacing](#) (self, ifactor=None, side=None)
- [setHighFullWell](#) (self, flag=True)
- [setZeroDeadTime](#) (self, flag=True, side=None)
- [setTriggerDelay](#) (self, delay=0)
- [setPhiDelay](#) (self, side=None, delay=0)
- [setExtClk](#) (self, dilation=None, frequency=None)
- [parseReadoff](#) (self, frames, columns=1)
- [sendCMD](#) (self, pkt)
- [arm](#) (self, mode=None)
- [readFrames](#) (self, waitOnSRAM=None, [timeout](#)=0, fast=False, columns=1)
- [readoff](#) (self, waitOnSRAM=None, [timeout](#)=0, fast=None, columns=1)
- [writeSerial](#) (self, cmd, [timeout](#)=None)
- [readSerial](#) (self, size, [timeout](#)=None)
- [closeDevice](#) (self)
- [saveTiffs](#) (self, frames, path=None, filename="Frame", prefix=None, index=None)
- [plotFrames](#) (self, frames, index=None)
- [getEnter](#) (self, text)
- [checkCRC](#) (self, rval)
- [str2bytes](#) (self, astring)
- [bytes2str](#) (self, bytesequenece)
- [str2npparray](#) (self, valstring)
- [flattenlist](#) (self, mylist)
- [partition](#) (self, frames, columns)
- [initialize](#) (self)
- *End aliases.*
- [reinitialize](#) (self)
- [reboot](#) (self)
- [getBoardInfo](#) (self)
- [getRegister](#) (self, regname)
- [setRegister](#) (self, regname, regval)
- [resolveSubreg](#) (self, srname)
- [getSubregister](#) (self, subregname)
- [setSubregister](#) (self, subregname, valstring)
- [submitMessages](#) (self, messages, errorstring="Error")
- [getPot](#) (self, potname, errflag=False)
- [setPot](#) (self, potname, value=1.0, errflag=False)
- [getPotV](#) (self, potname, errflag=False)
- [setPotV](#) (self, potname, voltage, tune=False, accuracy=0.01, iterations=20, approach=0.75, errflag=False)
- [getMonV](#) (self, monname, errflag=False)
- [readImgs](#) (self, waitOnSRAM=True, mode="Hardware")
- [saveFrames](#) (self, frames, path=None, filename="frames", prefix=None)
- [saveNumpys](#) (self, frames, path=None, filename="Frame", prefix=None, index=None)
- [dumpNumpy](#) (self, datastream, path=None, filename="Dump", prefix=None)

- [checkRegSet](#) (self, regname, teststring)
- [initPowerCheck](#) (self)
- [powerCheck](#) (self, delta=10)
- [printBoardInfo](#) (self)
- [dumpRegisters](#) (self)
- [dumpSubregisters](#) (self)
- [mmReadoff](#) (self, waitOnSRAM, variation=None)
- [setFrames](#) (self, minframe=None, maxframe=None)
- [setRows](#) (self, minrow=0, maxrow=None, [padToFull](#)=False)
- [abortReadoff](#) (self, flag=True)
- [batchAcquire](#) (self, sets=1, trig="Hardware", path=None, filename="Frame", prefix=None, showProgress=0)
- [saveHDF](#) (self, frames, path=None, filename="Acquisition", prefix=None)

Public Attributes

- [version](#)
- [currtime](#)
- [oldtime](#)
- [trigtime](#)
- [waited](#)
- [read](#)
- [unstringed](#)
- [parsedtime](#)
- [savetime](#)
- [cycle](#)
- [boardname](#)
- [timeout](#)
- [commname](#)
- [sensorname](#)

For regular version.

- [verbose](#)
- [port](#)
- [python](#)
- [pyth1](#)
- [pyth2](#)
- [PY3](#)
- [platform](#)
- [arch](#)
- [FPGAVersion](#)
- [FPGANum](#)
- [FPGAboardtype](#)
- [FPGArad](#)
- [FPGAsensor](#)
- [FPGAinterfaces](#)
- [FPGAinvalid](#)
- [iplist](#)
- [packageroot](#)
- [armed](#)
- [sens timing](#)
- [sensmanual](#)

- [inittime](#)
- [padToFull](#)
- [abort](#)
- [verbmap](#)
- [logtag](#)
- [logcritbase](#)
- [logerrbase](#)
- [logwarnbase](#)
- [loginfobase](#)
- [logdebugbase](#)
- [logcrit](#)
- [logerr](#)
- [logwarn](#)
- [loginfo](#)
- [logdebug](#)
- [verblevel](#)
- [payloaderror](#)
- [sensor](#)
- [comms](#)
- [board](#)

6.1.1 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
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
saveFrames(frames) - save image object as one file
saveNumpys(frames) - save individual frames as numpy data files
dumpNumpy(datastream) - save datastream string to numpy file
powerCheck(delta) - check that board power has not failed
printBoardInfo() - print board information derived from FPGA_NUM register
dumpRegisters() - return contents of all board registers
dumpSubregisters() - return contents of all named board subregisters
setFrames(min, max) - select subset of frames for readoff
setRows(min, max, padToFull) - select subset of rows for readoff
abortReadoff() - cancel readoff in wait-for-SRAM loop
batchAcquire() - 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 methods:

Board methods

disarm() - take camera out of waiting-for-trigger state
 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)
 getTemp() - read on-board temperature sensor
 getPressure() - read on-board pressure sensor
 dumpStatus() - generate dictionary of status, register, and subregister contents

Sensor methods

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
 setManualTiming() - configures manual shutter timing
 getManualTiming() - returns manual shutter settings from registers
 selectOscillator(osc) - select timing oscillator
 setInterlacing(ifactor) - sets interlacing factor
 setHighFullWell(flag) - controls High Full Well mode
 setZeroDeadTime(flag, side) - controls Zero Dead Time mode
 setTriggerDelay(delayblocks) - sets trigger delay

Comms methods

sendCMD(pkt)- sends packet object via serial port
 arm() - configures software buffers & arms camera
 readFrames() - waits for data ready flag, then downloads image data
 readoff() - waits for data ready flag, then downloads image data
 closeDevice() - disconnect interface and release resources

Miscellaneous functions (bare functions that can be called as methods)

saveTiffs(frames) - save individual frames as tiffs
 plotFrames(frames) - plot individual frames as tiffs

Informational class variables:

version - nsCamera software version
 FPGAVersion - firmware version (date)
 FPGAEnum - firmware implementation identifier
 FPGABoardtype - FPGA self-identified board type (should match 'boardname')
 FPGArad = Flag indicating radiation-tolerant FPGA build
 FPGAAsensor = FPGA self-identified sensor family (should correspond to 'sensorname')
 FPGAinterfaces = FPGA self-identified interfaces (list should include 'commname')
 FPGAinvalid = flag indicating invalid FPGA information in register

Definition at line 59 of file [CameraAssembler.py](#).

6.1.2 Constructor & Destructor Documentation

6.1.2.1 __init__()

```
nsCamera.CameraAssembler.CameraAssembler.__init__(
    self,
```

```

boardname = "llnl_v4",
commname = "GigE",
sensorname = "icarus2",
verbose = 4,
port = None,
ip = None,
logfile = None,
logtag = None,
timeout = 30 )

```

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)
    5. print DEBUG logging messages (detailed messages describing specific
      operations and messages)
port: optional integer
    When using RS422, this preselects the comport for RS422 and bypasses
      port search
    When using GigE, this preselects the OrangeTree control port for GigE
      (ignored if an ip parameter is not also provided)
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
timeout: timeout in seconds for connecting using Gigabit Ethernet

```

Definition at line 151 of file [CameraAssembler.py](#).

```

00162     ):
00163         """
00164         Args:
00165             boardname: name of FPGA board: llnl_v1, llnl_v4
00166             commname: name of communication interface: rs422, gige
00167             sensorname: name of sensor: icarus, icarus2, daedalus
00168             verbose: optional, sets logging level
00169                 0: print no logging messages
00170                 1: print CRITICAL logging messages (camera will not operate, e.g.,
00171                   unable to connect to board)
00172                 2: print ERROR logging messages (camera will not operate as directed,
00173                   e.g., an attempt to set the timing mode has failed, but the camera
00174                   is still operational)
00175                 3: print WARNING logging messages (camera will operate as directed, but
00176                   perhaps not as expected, e.g., ca.setTiming('A', (9, 8), 1) may be
00177                   programmed correctly, but the actual timing generated by the board
00178                   will be {1} [9, 8, 9, 14, 9, 8, 9])
00179                 4: print INFO logging messages (operational messages from ordinary
00180                   camera operation)
00181                 5. print DEBUG logging messages (detailed messages describing specific
00182                   operations and messages)
00183             port: optional integer
00184                 When using RS422, this preselects the comport for RS422 and bypasses
00185                   port search
00186                 When using GigE, this preselects the OrangeTree control port for GigE
00187                   (ignored if an ip parameter is not also provided)
00188             ip: optional string (e.g., '192.168.1.100')

```

```

00189         GigE: bypasses network search and selects particular OrangeTree board -
00190             required for some operating systems
00191         logfile: optional string, name of file to divert console output
00192         timeout: timeout in seconds for connecting using Gigabit Ethernet
00193     """
00194     self.version = "2.1.2"
00195     self.currttime = 0
00196     self.OLDtime = 0
00197     self.trigtime = []
00198     self.waited = []
00199     self.read = []
00200     self.unstringed = []
00201     self.parsedtime = []
00202     self.savetime = []
00203     self.cycle = []
00204     self.boardname = boardname.lower()
00205     self.timeout = timeout
00206     # TODO: parse boardname, etc. in separate method
00207     if self.boardname in ["1lnlv1", "v1", "1", 1]:
00208         self.boardname = "1lnlv1"
00209     if self.boardname in ["1lnlv4", "v4", "4", 4]:
00210         self.boardname = "1lnlv4"
00211     self.commname = commname.lower()
00212     if self.commname[0] == "g" or self.commname[0] == "e":
00213         self.commname = "gige"
00214     if self.commname[0] == "r":
00215         self.commname = "rs422"
00216     self.sensorname = sensorname.lower()
00217     if self.sensorname in ["i1", "ic1", "icarus1"]:
00218         self.sensorname = "icarus"
00219     if self.sensorname in ["i2", "ic2"]:
00220         self.sensorname = "icarus2"
00221     if self.sensorname == "d":
00222         self.sensorname = "daedalus"
00223     self.verbose = int(verbose)
00224     self.port = port
00225     self.python, self.pyth1, self.pyth2, _, _ = sys.version_info
00226     self.PY3 = self.python >= 3
00227     self.platform = platform.system()
00228     self.arch, _ = platform.architecture()
00229
00230     self.FPGAVersion = ""
00231     self.FPGANum = ""
00232     # FPGA information here and below populated during initialization using
00233     #   getBoardInfo
00234     self.FPGAboardtype = ""
00235     self.FPGArad = False
00236     self.FPGAAsensor = ""
00237     self.FPGAinterfaces = []
00238
00239     # indicates invalid FPGA information in register# (0x80000001 accepted as valid)
00240     self.FPGAinvalid = False
00241
00242     self.iplist = None
00243     self.packageroot = os.path.dirname(inspect.getfile(CameraAssembler))
00244     self.armed = False
00245
00246     # only one of these collections (sensstming, sensmanual) should be nonempty at
00247     #   any given time
00248     self.sensstming = {} # preserve HST setting against possible power failure
00249     self.sensmanual = [] # preserve manual timing
00250     self.inittime = 0
00251     self.padToFull = False
00252     self.abort = False
00253
00254     self.verbmap = {
00255         0: 99,
00256         1: logging.CRITICAL,
00257         2: logging.ERROR,
00258         3: logging.WARNING,
00259         4: logging.INFO,
00260         5: logging.DEBUG,
00261     }
00262     if logtag is None:
00263         logtag = ""
00264     self.logtag = logtag
00265
00266     self.logcritbase = "CRITICAL {logtag}: ".format(logtag=logtag)
00267     self.logerrbase = "ERROR {logtag}: ".format(logtag=logtag)
00268     self.logwarnbase = "WARNING {logtag}: ".format(logtag=logtag)
00269     self.loginfobase = "INFO {logtag}: ".format(logtag=logtag)

```

```

00270         self.logdebugbase = "DEBUG {logtag}:".format(logtag=logtag)
00271
00272         self.logcrit = "{lb}[CA]".format(lb=self.logcritbase)
00273         self.logerr = "{lb}[CA]".format(lb=self.logerrbase)
00274         self.logwarn = "{lb}[CA]".format(lb=self.logwarnbase)
00275         self.loginfo = "{lb}[CA]".format(lb=self.loginfobase)
00276         self.logdebug = "{lb}[CA]".format(lb=self.logdebugbase)
00277
00278         self.verblevel = self.verbmap.get(verbose, 5) # defaults to 5 for invalid entry
00279
00280         if logfile:
00281             logging.basicConfig(format="%(message)s", filename=logfile)
00282         else:
00283             logging.basicConfig(format="%(message)s")
00284             logging.getLogger().setLevel(self.verblevel)
00285             logging.getLogger("matplotlib.font_manager").disabled = True
00286             logging.debug(
00287                 "{logdebug}CameraAssembler: boardname = {boardname}; commname = {commname};"
00288                 " sensorname = {sensorname}; verbose = {verbose}; port = {port}; ip = {ip};"
00289                 " logfile = {logfile}; logtag = {logtag}:".format(
00290                     logdebug=self.logdebug,
00291                     boardname=boardname,
00292                     commname=commname,
00293                     sensorname=sensorname,
00294                     verbose=verbose,
00295                     port=port,
00296                     ip=ip,
00297                     logfile=logfile,
00298                     logtag=logtag,
00299             )
00300         )
00301
00302         if ip:
00303             try:
00304                 iphex = socket.inet_aton(ip)
00305             except socket.error:
00306                 logging.critical(
00307                     "{logcrit}CameraAssembler: invalid IP provided".format(
00308                         logcrit=self.logcrit
00309                     )
00310                 )
00311                 sys.exit(1)
00312             ipnum = [0, 0, 0, 0]
00313             for i in range(4):
00314                 if self.PY3:
00315                     ipnum[i] = iphex[i]
00316                 else:
00317                     ipnum[i] = int(iphex[i].encode("hex"), 16)
00318             self.iplist = ipnum
00319
00320         self.payloaderror = False
00321
00322         # code pulled out of __init__ to facilitate reinitialization of the board
00323         # without needing to instantiate a new CameraAssembler object
00324         self.initialize()
00325

```

6.1.3 Member Function Documentation

6.1.3.1 abortReadoff()

```

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 1989 of file [CameraAssembler.py](#).

```
01989     def abortReadoff(self, flag=True):
01990         """
01991         Simple abort command for readoff in waiting mode--does not interrupt download in
01992         progress. Requires external threading to function. *WARNING* if not
01993         intercepted by active readoff command, will terminate next readoff command
01994         immediately at inception.
01995         Args:
01996             flag: Sets passive abort flag read by readoff command
01997         Returns:
01998             boolean: updated setting of flag
01999         """
02000         logging.info(self.loginfo + "abortReadoff")
02001         self.abort = flag
02002         return flag
02003
```

6.1.3.2 arm()

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

Definition at line 452 of file [CameraAssembler.py](#).

```
00452     def arm(self, mode=None):
00453         return self.comms.arm(mode)
00454
```

6.1.3.3 batchAcquire()

```
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.
WARNING This method stores images in RAM, so the number of sets that can be acquired in a single call is limited by available memory.

Args:

```
sets: Number of acquisitions to perform
trig: trigger type; 'hardware', 'software', or 'dual'
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 2004 of file [CameraAssembler.py](#).

```

02012     ):
02013         """
02014         Acquire a series of images as fast as possible, then process and save to disk.
02015         *WARNING* This method stores images in RAM, so the number of sets that can be
02016         acquired in a single call is limited by available memory.
02017
02018         Args:
02019             sets: Number of acquisitions to perform
02020             trig: trigger type; 'hardware', 'software', or 'dual'
02021             path: save path, defaults to './output'
02022             filename: defaults to 'frames.bin'
02023             prefix: prepended to filename, defaults to time/date (e.g. '160830-124704_')
02024                     DO NOT USE unless providing a varying value (a fixed prefix will cause
02025                     overwriting)
02026             showProgress: if non-zero, show notice every 'showProgress' acquisitions and
02027                           print total acquisition time
02028
02029         Returns:
02030             Time taken for acquisition (seconds)
02031         """
02032         logging.debug(
02033             self.logdebug
02034             + "batchAcquire: sets = "
02035             + str(sets)
02036             + "; trig = "
02037             + str(trig)
02038             + "; path = "
02039             + str(path)
02040             + "; filename = "
02041             + str(filename)
02042             + "; prefix = "
02043             + str(prefix)
02044             + "; showProgress = "
02045             + str(showProgress)
02046         )
02047         datalist = ["0"] * sets
02048         timelist = [datetime.now()] * sets
02049         logging.info(
02050             self.loginfo
02051             + "batchAcquire: temporarily disabling warning and information logging "
02052         )
02053         logging.getLogger().setLevel(self.verbmap.get(2))
02054         beforeread = time.time()
02055         for i in range(sets):
02056             if showProgress and not (i + 1) % showProgress:
02057                 print(self.loginfo + "batchAcquire: Acquiring set " + str(i + 1))
02058             self.arm(trig)
02059             data, datalen, data_err = self.readoff(fast=True)
02060             datalist[i] = data
02061             timelist[i] = datetime.now()
02062         afterread = time.time()
02063         if showProgress:
02064             print(
02065                 self.loginfo
02066                 + "batchAcquire: "
02067                 + str(afterread - beforeread)
02068                 + " seconds for "
02069                 + str(sets)
02070                 + " sets"
02071             )
02072         setnum = 0
02073         if path is None:
02074             path = os.path.join(os.getcwd(), "output")
02075         for imset, imtime in zip(datalist, timelist):
02076             setnum = setnum + 1
02077             if showProgress and not setnum % showProgress:
02078                 print(self.loginfo + "batchAcquire: Saving set " + str(setnum))
02079             parsed = generateFrames(self, imset)
02080             if prefix is None:
02081                 setprefix = imtime.strftime("%Y%m%d-%H%M%S%f")[:-2] + "_"
02082             else:
02083                 setprefix = prefix
02084             self.saveTiffs(parsed, path, filename, prefix=setprefix)
02085         logging.getLogger().setLevel(self.verblevel)
02086         logging.info(self.loginfo + "batchAcquire: re-enabling logging")
02087         return afterread - beforeread
02088

```


6.1.3.4 bytes2str()

```
nsCamera.CameraAssembler.CameraAssembler.bytes2str (
    self,
    bytesequenece )
```

Definition at line 486 of file [CameraAssembler.py](#).

```
00486     def bytes2str(self, bytesequenece):
00487         return bytes2str(bytesequenece)
00488
```

6.1.3.5 checkCRC()

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

Definition at line 480 of file [CameraAssembler.py](#).

```
00480     def checkCRC(self, rval):
00481         return checkCRC(rval)
00482
```

6.1.3.6 checkRegSet()

```
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 integer or hexadecimal string
               with or without '0x'
```

Returns:

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

Definition at line 1662 of file [CameraAssembler.py](#).

```
01662     def checkRegSet(self, regname, teststring):
01663         """
01664         Quick check to confirm that data read from register matches data write
01665
01666         Args:
01667             regname: register to test
01668             teststring: value to assign to register, as integer or hexadecimal string
01669                       with or without '0x'
01670
01671         Returns:
01672             boolean, True if read and write values match
01673         """
01674         self.setRegister(regname, teststring)
01675         # tell board to send data; wait to clear before interrogating register contents
01676         if regname == "SRAM_CTL":
01677             time.sleep(2)
01678             if self.comname == "rs422":
01679                 logging.info(
01680                     self.loginfo + "skipping 'SRAM_CTL' register check for RS422"
```

```

01681         )
01682         return True
01683     else:
01684         time.sleep(0.1)
01685         temp = self.getRegister(regname)
01686         resp = temp[1].upper()
01687         if resp != teststring.upper():
01688             logging.error(
01689                 self.logerr
01690                 + "checkRegSet failure: "
01691                 + regname
01692                 + " ; set: "
01693                 + teststring
01694                 + " ; read: "
01695                 + resp
01696             )
01697         return False
01698     return True
01699

```

6.1.3.7 checkSensorVoltStat()

```

nsCamera.CameraAssembler.CameraAssembler.checkSensorVoltStat (
    self )

```

Definition at line 398 of file [CameraAssembler.py](#).

```

00398     def checkSensorVoltStat(self):
00399         return self.sensor.checkSensorVoltStat()
00400

```

6.1.3.8 checkStatus()

```

nsCamera.CameraAssembler.CameraAssembler.checkStatus (
    self )

```

Definition at line 383 of file [CameraAssembler.py](#).

```

00383     def checkStatus(self):
00384         return self.board.checkStatus()
00385

```

6.1.3.9 checkStatus2()

```

nsCamera.CameraAssembler.CameraAssembler.checkStatus2 (
    self )

```

Definition at line 386 of file [CameraAssembler.py](#).

```

00386     def checkStatus2(self):
00387         return self.board.checkStatus2()
00388

```

6.1.3.10 clearStatus()

```

nsCamera.CameraAssembler.CameraAssembler.clearStatus (
    self )

```

Definition at line 380 of file [CameraAssembler.py](#).

```

00380     def clearStatus(self):
00381         return self.board.clearStatus()
00382

```

6.1.3.11 closeDevice()

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

Definition at line 468 of file [CameraAssembler.py](#).

```
00468     def closeDevice(self):
00469         return self.comms.closeDevice()
00470
```

6.1.3.12 configADCs()

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

Definition at line 341 of file [CameraAssembler.py](#).

```
00341     def configADCs(self):
00342         return self.board.configADCs()
00343
```

6.1.3.13 disarm()

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

Definition at line 344 of file [CameraAssembler.py](#).

```
00344     def disarm(self):
00345         return self.board.disarm()
00346
```

6.1.3.14 dumpNumpy()

```
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. The packet header is removed before saving

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 1610 of file [CameraAssembler.py](#).

```

01616     ):
01617         """
01618         Datastream is converted directly to numpy array and saved to disk. No attempt to
01619         parse headers or separate into individual frames is made. The packet header is
01620         removed before saving
01621
01622         Args:
01623             datastream: string to be saved
01624             path: save path, defaults to './output'
01625             filename: defaults to 'Dump'
01626             prefix: prepended to 'filename', defaults to time/date
01627                    (e.g. '160830-124704_')
01628
01629         Returns:
01630             Error string
01631         """
01632         logging.info(self.logininfo + "dumpNumpy")
01633         logging.debug(
01634             self.logdebug
01635             + "dumpNumpy: path = "
01636             + str(path)
01637             + "; filename = "
01638             + str(filename)
01639             + "; prefix = "
01640             + str(prefix)
01641         )
01642         err = ""
01643         if path is None:
01644             path = os.path.join(os.getcwd(), "output")
01645         if prefix is None:
01646             prefix = time.strftime("%y%m%d-%H%M%S_", time.localtime())
01647         if not os.path.exists(path):
01648             os.makedirs(path)
01649         npdata = str2nparray(datastream[36:])
01650         try:
01651             nppath = os.path.join(path, prefix + filename + ".np")
01652             np.save(nppath, npdata)
01653         except SystemExit:
01654             raise
01655         except KeyboardInterrupt:
01656             raise
01657         except Exception:
01658             err = self.logerr + "dumpNumpy: unable to save data stream"
01659             logging.error(err)
01660         return err
01661

```

6.1.3.15 dumpRegisters()

```

nsCamera.CameraAssembler.CameraAssembler.dumpRegisters (
    self )

```

DEPRECATED use dumpStatus() instead

List contents of all registers in board.registers. **WARNING** some status flags will reset when read.

Returns:

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

Definition at line 1764 of file [CameraAssembler.py](#).

```

01764     def dumpRegisters(self):
01765         """
01766         *DEPRECATED* use dumpStatus() instead
01767
01768         List contents of all registers in board.registers. *WARNING* some status flags
01769         will reset when read.
01770
01771         Returns:

```

```

01772         Sorted list: [register name (register address) : register contents as
01773                       hexadecimal string without '0x']
01774         """
01775         dump = {}
01776         for key in self.board.registers.keys():
01777             err, rval = self.getRegister(key)
01778             dump[key] = rval
01779         reglistmax = int(max(self.board.registers.values()), 16)
01780         dumplist = [0] * (reglistmax + 1)
01781         for k, v in dump.items():
01782             regnum = self.board.registers[k]
01783             dumplist[int(regnum, 16)] = (
01784                 "(" + regnum + ") {0:<24} {1}".format(k, v.upper())
01785             )
01786         reglist = [a for a in dumplist if a]
01787         return reglist
01788

```

6.1.3.16 dumpStatus()

```

nsCamera.CameraAssembler.CameraAssembler.dumpStatus (
    self )

```

Definition at line 395 of file [CameraAssembler.py](#).

```

00395     def dumpStatus(self):
00396         return self.board.dumpStatus()
00397

```

6.1.3.17 dumpSubregisters()

```

nsCamera.CameraAssembler.CameraAssembler.dumpSubregisters (
    self )

```

DEPRECATED use dumpStatus() instead

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

Returns:

```

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

Definition at line 1789 of file [CameraAssembler.py](#).

```

01789     def dumpSubregisters(self):
01790         """
01791         *DEPRECATED* use dumpStatus() instead
01792
01793         List contents of all subregisters in board.channel_lookups and
01794         board.monitor_lookups.
01795         *WARNING* some registers will reset when read; only the first subregister from
01796         such a register will return the correct value, the remainder will return zeros
01797
01798         Returns:
01799             dictionary {subregister name : subregister contents as binary string
01800                       without initial '0b'}
01801         """
01802         dump = {}
01803         for sub in self.board.subreglist:
01804             key = sub.name
01805             err, resp = self.getSubregister(key)
01806             if err:
01807                 logging.warning(
01808                     self.logwarn + "dumpSubregisters: unable to read subregister " + key
01809                 )
01810             val = hex(int(resp, 2))
01811             dump[key] = val
01812         return dump
01813

```

6.1.3.18 enableLED()

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

Definition at line 362 of file [CameraAssembler.py](#).

```
00362     def enableLED(self, status=1):
00363         return self.board.enableLED(status)
00364
```

6.1.3.19 flattenlist()

```
nsCamera.CameraAssembler.CameraAssembler.flattenlist (
    self,
    mylist )
```

Definition at line 492 of file [CameraAssembler.py](#).

```
00492     def flattenlist(self, mylist):
00493         return flattenlist(mylist)
00494
```

6.1.3.20 getBoardInfo()

```
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 643 of file [CameraAssembler.py](#).

```
00643     def getBoardInfo(self):
00644         """
00645         Get board info from FPGA_NUM register. Returns error flag if register contents
00646         are invalid and tuple (board version number, rad tolerance flag, sensor name)
00647
00648         Returns:
00649             tuple (errorFlag, (board version, rad tolerance flag, sensor name))
00650         """
00651         invalidFPGANum = False
00652         interfaces = []
00653
00654         # TODO: move to new method (combine with parsing from initialize)
00655         if int(self.FPGANum[0], 16) & 8:
00656             if self.FPGANum[1] == "1":
00657                 boardtype = "LLNLv1"
00658             elif self.FPGANum[1] == "4":
00659                 boardtype = "LLNLv4"
00660             else:
00661                 boardtype = "LLNLv?"
00662                 invalidFPGANum = True
00663         else:
00664             boardtype = "SNLrevC"
00665             logging.warning(
00666                 self.logwarn + "FPGA self-identifies as SNLrevC, which is not "
00667                 "supported by this software "
00668             )
00669             invalidFPGANum = True
```

```

00670         self.FPGAboardtype = boardtype
00671
00672         if int(self.FPGANum[6], 16) & 1:
00673             rad = True
00674         else:
00675             rad = False
00676         self.FPGArad = rad
00677
00678         if self.FPGANum[7] == "1":
00679             sensor = "Icarus"
00680         elif self.FPGANum[7] == "2":
00681             sensor = "Daedalus"
00682         else:
00683             sensor = "Undefined"
00684             invalidFPGANum = True
00685         self.FPGAensor = sensor
00686
00687         if int(self.FPGANum[5], 16) & 1:
00688             interfaces.append("RS422")
00689         if int(self.FPGANum[5], 16) & 2:
00690             interfaces.append("GigE")
00691         self.FPGAinterfaces = interfaces
00692
00693         if invalidFPGANum:
00694             if self.FPGANum == "80000001":
00695                 invalidFPGANum = False
00696             else:
00697                 logging.warning(self.logwarn + "FPGA self-identification is invalid")
00698         self.FPGAinvalid = invalidFPGANum
00699
00700         return invalidFPGANum, (boardtype, rad, sensor)
00701

```

6.1.3.21 getEnter()

```

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

```

Definition at line 477 of file [CameraAssembler.py](#).

```

00477     def getEnter(self, text):
00478         return getEnter(text)
00479

```

6.1.3.22 getManualTiming()

```

nsCamera.CameraAssembler.CameraAssembler.getManualTiming (
    self )

```

Definition at line 416 of file [CameraAssembler.py](#).

```

00416     def getManualTiming(self):
00417         return self.sensor.getManualTiming()
00418

```

6.1.3.23 getMonV()

```

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 1396 of file [CameraAssembler.py](#).

```
01396     def getMonV(self, monname, errflag=False):
01397         """
01398         Reads voltage from monitor named or that associated with the pot named 'monname'
01399
01400         Args:
01401             monname: name of pot or monitor, e.g., VRST or MON_CH2 found in
01402             board.subreg_aliases or defined in board.subregisters
01403             errflag: if True, return tuple with error string
01404
01405         Returns:
01406             if errflag:
01407                 tuple: (error string, float value of voltage measured by monitor)
01408             else:
01409                 float value of voltage measured by monitor
01410         """
01411         logging.debug(
01412             self.logdebug
01413             + "getMonV: monname = "
01414             + str(monname)
01415             + "; errflag = "
01416             + str(errflag)
01417         )
01418         monname = monname.upper()
01419         if monname in self.board.subreg_aliases:
01420             monname = self.board.subreg_aliases[monname].upper()
01421         # else:
01422         for key, value in self.board.monitor_controls.items():
01423             if value == monname:
01424                 monname = key
01425         if monname not in self.board.monitor_controls:
01426             if monname in self.board.subreglist:
01427                 pass # no change necessary
01428             else:
01429                 err = (
01430                     self.logerr + "getMonV: invalid lookup " + monname + ", returning 0"
01431                 )
01432                 logging.error(err)
01433                 if errflag:
01434                     return err, 0
01435                 return 0
01436         err, monval = self.getPot(monname, errflag=True)
01437         logging.debug(self.logdebug + "getMonV: monval = " + str(monval))
01438         if err:
01439             logging.error(
01440                 self.logerr + "getMonV: unable to read monitor value for " + monname
01441             )
01442         if self.board.ADC5_bipolar:
01443             if monval >= 0.5:
01444                 monval -= 1 # handle negative measurements (two's complement)
01445             if errflag:
01446                 return err, 2 * self.board.ADC5_mult * monval * self.board.VREF
01447             return 2 * self.board.ADC5_mult * monval * self.board.VREF
01448         else:
01449             if errflag:
01450                 return err, self.board.ADC5_mult * monval * self.board.VREF
01451             return self.board.ADC5_mult * monval * self.board.VREF
01452
```

6.1.3.24 getPot()

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 1013 of file [CameraAssembler.py](#).

```

01013     def getPot(self, potname, errflag=False):
01014         """
01015         Retrieves value of pot or ADC monitor subregister, scaled to [0,1).
01016
01017         Args:
01018             potname: name of pot or monitor, e.g., VRST or MON_CH2 found in
01019                     board.subreg_aliases or defined in board.subregisters
01020             errflag: if True, return tuple with error string
01021
01022         Returns:
01023             if errflag:
01024                 tuple: (error string, float value of subregister, scaled to [0,1) )
01025             else:
01026                 float value of subregister, scaled to [0,1)
01027         """
01028         logging.debug(
01029             "{logdebug}getPot: potname = {potname}; errflag = {errflag}".format(
01030                 logdebug=self.logdebug, potname=potname, errflag=errflag
01031             )
01032         )
01033
01034         potname, potobj, _ = self.resolveSubreg(potname)
01035         if not potobj:
01036             err = "{logerr}getPot: invalid lookup: {potname}; returning 0".format(
01037                 logerr=self.logerr, potname=potname
01038             )
01039
01040             logging.error(err)
01041             if errflag:
01042                 return err, 0
01043             return 0
01044         err, b_pot_value = self.getSubregister(potname)
01045         if err:
01046             err = "{logerr}getPot: unable to read subregister: {potname}".format(
01047                 logerr=self.logerr, potname=potname
01048             )
01049
01050             # convert binary string back to decimal
01051             f_reg_value = 1.0 * int(b_pot_value, 2)
01052             value = (f_reg_value - potobj.min) / (potobj.max - potobj.min)
01053             # logging.debug(self.logdebug + "getpot: value = " + str(value))
01054
01055             logging.debug(
01056                 "{logdebug}getpot: value = {value}".format(
01057                     logdebug=self.logdebug, value=value
01058                 )
01059             )
01060
01061             if errflag:
01062                 return err, value
01063             return value
01064

```

6.1.3.25 getPotV()

```
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 1155 of file [CameraAssembler.py](#).

```
01155     def getPotV(self, potname, errflag=False):
01156         """
01157         Reads voltage _setting_ (not actual voltage) of specified pot
01158
01159         Args:
01160             potname: name of pot or monitor, e.g., VRST or MON_CH2 found in
01161             board.subreg_aliases or defined in board.subregisters
01162             errflag: if True, return tuple with error string
01163
01164         Returns:
01165             if errflag:
01166                 tuple: (error string, float value of pot voltage)
01167             else:
01168                 float value of pot voltage
01169         """
01170         logging.debug(
01171             self.logdebug
01172             + "getPotV: potname = "
01173             + str(potname)
01174             + "; errflag = "
01175             + str(errflag)
01176         )
01177         potname, potobj, _ = self.resolveSubreg(potname)
01178         if not potobj:
01179             err = (
01180                 self.logerr + "getPotV: invalid lookup: " + potname + " , returning 0 "
01181             )
01182             logging.error(err)
01183             if errflag:
01184                 return err, 0
01185             return 0
01186         err, val = self.getPot(potname, errflag=True)
01187         logging.debug(self.logdebug + "getPotV: val = " + str(val))
01188         if err:
01189             logging.error(self.logerr + "getPotV: unable to read pot " + potname)
01190         minV = potobj.minV
01191         maxV = potobj.maxV
01192         if errflag:
01193             return err, val * (maxV - minV)
01194         return val * (maxV - minV)
01195
```

6.1.3.26 getPressure()

```
nsCamera.CameraAssembler.CameraAssembler.getPressure (
    self,
```

```

        offset = None,
        sensitivity = None,
        units = None )

```

Definition at line 377 of file [CameraAssembler.py](#).

```

00377     def getPressure(self, offset=None, sensitivity=None, units=None):
00378         return self.board.getPressure(offset, sensitivity, units)
00379

```

6.1.3.27 getRegister()

```

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 702 of file [CameraAssembler.py](#).

```

00702     def getRegister(self, regname):
00703         """
00704         Retrieves contents of named register as hexadecimal string without '0x'
00705
00706         Args:
00707             regname: name of register as given in ICD
00708
00709         Returns:
00710             tuple: (error string, register contents as hexadecimal string without '0x')
00711         """
00712         # logging.debug(self.logdebug + "getRegister: regname = " + str(regname))
00713         logging.debug(
00714             "{logdebug}getRegister: regname = {regname}".format(
00715                 logdebug=self.logdebug, regname=regname
00716             )
00717         )
00718
00719         regname = regname.upper()
00720         if regname not in self.board.registers:
00721             err = "{logerr}getRegister: Invalid register name: {regname}; returning"
00722             " zeros".format(logerr=self.logerr, regname=regname)
00723             logging.error(err)
00724             return err, "00000000"
00725         sendpkt = Packet(cmd="1", addr=self.board.registers[regname])
00726         err, rval = self.comms.sendCMD(sendpkt)
00727         if err:
00728             logging.error(
00729                 "{logerr}getRegister: {regname}; {err}".format(
00730                     logerr=self.logerr, regname=regname, err=err
00731                 )
00732             )
00733
00734         retval = rval[8:16]
00735         logging.debug(
00736             "{logdebug}getRegister: retval = {retval}".format(
00737                 logdebug=self.logdebug, retval=retval
00738             )
00739         )
00740
00741         return err, retval
00742

```

6.1.3.28 getSensTemp()

```
nsCamera.CameraAssembler.CameraAssembler.getSensTemp (
    self,
    scale = None,
    offset = None,
    slope = None,
    dec = 1 )
```

Definition at line 419 of file [CameraAssembler.py](#).

```
00419     def getSensTemp(self, scale=None, offset=None, slope=None, dec=1):
00420         return self.sensor.getSensTemp(scale, offset, slope, dec)
00421
```

6.1.3.29 getSubregister()

```
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 836 of file [CameraAssembler.py](#).

```
00836     def getSubregister(self, subregname):
00837         """
00838         Returns substring of register identified in board attribute 'subregname'
00839
00840         Args:
00841             subregname: listed in board.subreg_aliases or defined in board.subregisters
00842
00843         Returns:
00844             tuple: (error string, contents of subregister as binary string without '0b')
00845         """
00846         logging.debug(
00847             "{logdebug}getSubregister: subregname = {subregname}".format(
00848                 logdebug=self.logdebug,
00849                 subregname=subregname,
00850             )
00851         )
00852
00853         subregname, subregobj, _ = self.resolveSubreg(subregname)
00854         if not subregobj:
00855             err = "{logerr}getSubregister: invalid lookup: {subregname}; returning"
00856                 " string of zeroes".format(logerr=self.logerr, subregname=subregname)
00857
00858             logging.error(err)
00859             return err, "".zfill(8)
00860         err, resp = self.getRegister(subregobj.register)
00861         if err:
00862             logging.error(
00863                 "{logerr}getSubregister: unable to retrieve register setting: \
00864                 {subregname}; returning '0' string".format(
00865                     logerr=self.logerr, subregname=subregname
00866                 )
00867             )
00868
00869             return err, "".zfill(8)
00870         hex_str = "0x" + resp # this should be a hexadecimalstring
00871         b_reg_value = "{0:0=32b}".format(int(hex_str, 16)) # convert to binary string
```

```

00872         # list indexing is reversed from bit string; the last bit of the string is at
00873         # index 0 in the list (thus bit 0 is at index 0)
00874         startindex = 31 - subregobj.start_bit
00875         retval = b_reg_value[startindex : startindex + subregobj.width]
00876         logging.debug(
00877             "{logdebug}getSubregister: retval = {retval}".format(
00878                 logdebug=self.logdebug, retval=retval
00879             )
00880         )
00881         return "", retval
00882

```

6.1.3.30 getTemp()

```

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

```

Definition at line 374 of file [CameraAssembler.py](#).

```

00374     def getTemp(self, scale=None):
00375         return self.board.getTemp(scale)
00376

```

6.1.3.31 getTimer()

```

nsCamera.CameraAssembler.CameraAssembler.getTimer (
    self )

```

Definition at line 356 of file [CameraAssembler.py](#).

```

00356     def getTimer(self):
00357         return self.board.getTimer()
00358

```

6.1.3.32 getTiming()

```

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

```

Definition at line 407 of file [CameraAssembler.py](#).

```

00407     def getTiming(self, side=None, actual=None):
00408         return self.sensor.getTiming(side, actual)
00409

```

6.1.3.33 initBoard()

```

nsCamera.CameraAssembler.CameraAssembler.initBoard (
    self )

```

Aliases to other objects' methods TODO: properly delegate these methods.

Definition at line 329 of file [CameraAssembler.py](#).

```

00329     def initBoard(self):
00330         return self.board.initBoard()
00331

```

6.1.3.34 initialize()

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

End aliases.

Initialize board registers and set pots

Definition at line 500 of file [CameraAssembler.py](#).

```
00500     def initialize(self):
00501         """
00502         Initialize board registers and set pots
00503         """
00504         # TODO: automate sensor and board selection from firmware info
00505
00506
00507
00508         # get sensor
00509         # TODO: pull sensor, board, comm id out to separate methods
00510         if self.sensorname == "icarus":
00511             import nsCamera.sensors.icarus as snsr
00512         elif self.sensorname == "icarus2":
00513             import nsCamera.sensors.icarus2 as snsr
00514         elif self.sensorname == "daedalus":
00515             import nsCamera.sensors.daedalus as snsr
00516         else: # catch-all for added sensors to attempt object encapsulation
00517             sensormodname = ".sensors." + self.sensorname
00518             try:
00519                 sensormod = importlib.import_module(sensormodname, "nsCamera")
00520             except ImportError:
00521                 logging.critical(self.logcrit + "invalid sensor name")
00522                 sys.exit(1)
00523             snsr = getattr(sensormod, self.sensorname)
00524         self.sensor = snsr(self)
00525
00526         # kill existing connections (for reinitialize)
00527         if hasattr(self, "comms"):
00528             self.closeDevice()
00529
00530         # get communications interface
00531         if self.commname == "rs422":
00532             import nsCamera.comms.RS422 as comms
00533         elif self.commname == "gige":
00534             import nsCamera.comms.GigE as comms
00535         else:
00536             commsmodname = ".comms." + self.commname
00537             try:
00538                 commsmod = importlib.import_module(commsmodname, "nsCamera")
00539             except ImportError:
00540                 logging.critical(self.logcrit + "invalid comms name")
00541                 sys.exit(1)
00542             comms = getattr(commsmod, self.commname)
00543         self.comms = comms(self)
00544
00545         # get board
00546         if self.boardname == "llnl_v1":
00547             import nsCamera.boards.LLNL_v1 as brd
00548
00549             self.board = brd.llnl_v1(self)
00550         elif self.boardname == "llnl_v4":
00551             import nsCamera.boards.LLNL_v4 as brd
00552
00553             self.board = brd.llnl_v4(self)
00554         else:
00555             boardmodname = ".board." + self.boardname
00556             try:
00557                 boardmod = importlib.import_module(boardmodname, "nsCamera")
00558             except ImportError:
00559                 logging.critical(self.logcrit + "invalid board name")
00560                 sys.exit(1)
00561             boardobj = getattr(boardmod, self.boardname)
00562             self.board = boardobj(self)
00563
00564         # Now that board exists, initialize board-specific aliases for sensors
```

```

00565         self.sensor.init_board_specific()
00566
00567
00568
00569         # TODO: make cython the standard version
00570         # #####
00571         # # For cython version
00572         #
00573         # # get sensor
00574         # if self.sensorname == "icarus":
00575         #     import nsCamera.sensors.icarus as snsr
00576         #     self.sensor = snsr.icarus(self)
00577         # elif self.sensorname == "icarus2":
00578         #     import nsCamera.sensors.icarus2 as snsr
00579         #     self.sensor = snsr.icarus2(self)
00580         # elif self.sensorname == "daedalus":
00581         #     import nsCamera.sensors.daedalus as snsr
00582         #     self.sensor = snsr.daedalus(self)
00583         #
00584         # # kill existing connections (for reinitialize)
00585         # if hasattr(self, "comms"):
00586         #     self.closeDevice()
00587         #
00588         # # get communications interface
00589         # if self.commname == "rs422":
00590         #     import nsCamera.comms.RS422 as comms
00591         #     self.comms = comms.RS422(self)
00592         # elif self.commname == "gige":
00593         #     import nsCamera.comms.GigE as comms
00594         #     self.comms = comms.GigE(self)
00595         #
00596         # # get board
00597         # if self.boardname == "llnl_v1":
00598         #     import nsCamera.boards.LLNL_v1 as brd
00599         #     self.board = brd.llnl_v1(self)
00600         # elif self.boardname == "llnl_v4":
00601         #     import nsCamera.boards.LLNL_v4 as brd
00602         #     self.board = brd.llnl_v4(self)
00603         # #####
00604
00605         err, rval = self.getRegister("FPGA_NUM")
00606         if err or rval == "":
00607             err, rval = self.getRegister("FPGA_NUM")
00608             if err or rval == "":
00609                 logging.critical(
00610                     self.logcrit + "Initialization failed: unable to communicate with"
00611                     " board. "
00612                 )
00613                 sys.exit(1)
00614
00615         self.initBoard()
00616         self.initPots()
00617         self.initSensor()
00618         self.initPowerCheck()
00619         self.getBoardInfo()
00620         self.printBoardInfo()
00621

```

6.1.3.35 initPots()

```

nsCamera.CameraAssembler.CameraAssembler.initPots (
    self )

```

Definition at line 332 of file [CameraAssembler.py](#).

```

00332     def initPots(self):
00333         return self.board.initPots()
00334

```

6.1.3.36 initPowerCheck()

```

nsCamera.CameraAssembler.CameraAssembler.initPowerCheck (
    self )

```

Reset software and board timers for monitoring power status

Definition at line 1700 of file [CameraAssembler.py](#).

```
01700     def initPowerCheck(self):
01701         """
01702         Reset software and board timers for monitoring power status
01703         """
01704         self.inittime = time.time()
01705         logging.info(self.loginfo + "resetting timer for power check function")
01706         self.resetTimer()
01707
```

6.1.3.37 initSensor()

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

Definition at line 338 of file [CameraAssembler.py](#).

```
00338     def initSensor(self):
00339         return self.board.initSensor()
00340
```

6.1.3.38 latchPots()

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

Definition at line 335 of file [CameraAssembler.py](#).

```
00335     def latchPots(self):
00336         return self.board.latchPots()
00337
```

6.1.3.39 mmReadoff()

```
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 1814 of file [CameraAssembler.py](#).

```

01814     def mmReadoff(self, waitOnSRAM, variation=None):
01815         """
01816         Convenience function for parsing frames for use by MicroManager plugin
01817         Args:
01818             waitOnSRAM: readoff wait flag
01819             variation: format of frames generated from readoff
01820                 default - return first frame only
01821                 "LastFrame" - return last frame only
01822                 "Average" - provide average of frames as single frame
01823                 "Landscape" - stitch frames together horizontally into single wide frame
01824
01825         Returns:
01826             ndarray - single image frame
01827         """
01828         frames, datalen, data_err = self.readoff(waitOnSRAM)
01829         if variation == "LastFrame":
01830             return frames[self.sensor.nframes - 1]
01831         elif variation == "Average":
01832             return np.sum(frames, axis=0) // self.sensor.nframes
01833         elif variation == "Landscape":
01834             shaped = [
01835                 np.reshape(frame, (self.sensor.maxheight, self.sensor.maxwidth))
01836                 for frame in frames
01837             ]
01838             return np.concatenate(shaped, axis=1)
01839         else:
01840             return frames[0]
01841

```

6.1.3.40 parseReadoff()

```

nsCamera.CameraAssembler.CameraAssembler.parseReadoff (
    self,
    frames,
    columns = 1 )

```

Definition at line 446 of file [CameraAssembler.py](#).

```

00446     def parseReadoff(self, frames, columns=1):
00447         return self.sensor.parseReadoff(frames, columns)
00448

```

6.1.3.41 partition()

```

nsCamera.CameraAssembler.CameraAssembler.partition (
    self,
    frames,
    columns )

```

Definition at line 495 of file [CameraAssembler.py](#).

```

00495     def partition(self, frames, columns):
00496         return partition(self, frames, columns)
00497

```

6.1.3.42 plotFrames()

```

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

```

Definition at line 474 of file [CameraAssembler.py](#).

```

00474     def plotFrames(self, frames, index=None):
00475         return plotFrames(self, frames, index)
00476

```

6.1.3.43 powerCheck()

```
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 the 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 1708 of file [CameraAssembler.py](#).

```
01708     def powerCheck(self, delta=10):
01709         """
01710         Check to see if board power has persisted since powerCheck was last initialized.
01711         Compares time elapsed since initialization against board's timer. If the
01712         difference is greater than 'delta,' flag as False (power has likely failed)
01713
01714         Args:
01715             delta: difference in seconds permitted between software and board timers
01716
01717         Returns:
01718             boolean, 'True' means timer difference is less than 'delta' parameter;
01719             'False' indicates power failure
01720         """
01721         elapsed = time.time() - self.inittime
01722         logging.debug(self.logdebug + "powerCheck: elapsed time = " + str(elapsed))
01723         difference = abs(elapsed - self.getTimer())
01724         if difference > delta:
01725             logging.warning(
01726                 self.logwarn + "powerCheck function has failed; may indicate current "
01727                 "or recent power failure "
01728             )
01729         return difference < delta
01730
```

6.1.3.44 printBoardInfo()

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

Definition at line 1731 of file [CameraAssembler.py](#).

```
01731     def printBoardInfo(self):
01732         # TODO: add override option if logging level is above info
01733         logging.info(
01734             self.loginfo
01735             + "Python version: "
01736             + str(self.python)
01737             + "."
01738             + str(self.pyth1)
01739             + "."
01740             + str(self.pyth2)
01741         )
01742         logging.info(self.loginfo + "nsCamera software version: " + self.version)
01743         logging.info(self.loginfo + "FPGA firmware version: " + self.FPGAVersion)
01744         logging.info(self.loginfo + "FPGA implementation: " + self.FPGAEnum)
01745         if self.FPGAInvalid:
01746             logging.info(self.loginfo + "FPGA information unavailable")
01747         else:
01748             logging.info(self.loginfo + "Board type: " + self.FPGABoardtype)
```

```

01749         logging.info(self.loginfo + "Rad-Tolerant: " + str(self.FPGArad))
01750         logging.info(self.loginfo + "Sensor family: " + self.FPGAsensor)
01751         logging.info(self.loginfo + "Sensor label: " + self.sensor.loglabel)
01752         logging.info(
01753             self.loginfo + "Available interfaces: " + ", ".join(self.FPGAinterfaces)
01754         )
01755         if self.commname == "gige":
01756             ci = self.comms.CardInfoP.contents
01757             ip = ".".join(str(e) for e in [b for b in ci.IPAddr])
01758             logging.info(
01759                 self.loginfo + "GigE connected to " + ip + ":" + str(self.port)
01760             )
01761         elif self.commname == "rs422":
01762             logging.info(self.loginfo + "RS422 connected to " + self.comms.port)
01763

```

6.1.3.45 readFrames()

```

nsCamera.CameraAssembler.CameraAssembler.readFrames (
    self,
    waitOnSRAM = None,
    timeout = 0,
    fast = False,
    columns = 1 )

```

Definition at line 455 of file [CameraAssembler.py](#).

```

00455     def readFrames(self, waitOnSRAM=None, timeout=0, fast=False, columns=1):
00456         frames, _, _ = self.comms.readoff(waitOnSRAM, timeout, fast, columns)
00457         return frames
00458

```

6.1.3.46 readImgs()

```

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 1453 of file [CameraAssembler.py](#).

```

01453     def readImgs(self, waitOnSRAM=True, mode="Hardware"):
01454         """
01455         Combines arm() and readoff() functions
01456
01457         Returns:
01458             tuple (list of numpy arrays, length of downloaded payload, payload error
01459                 flag) returned by readoff
01460         """
01461         logging.info(self.loginfo + "readImgs")
01462         self.arm(mode)
01463         return self.readoff(waitOnSRAM)
01464

```

6.1.3.47 readoff()

```
nsCamera.CameraAssembler.CameraAssembler.readoff (
    self,
    waitOnSRAM = None,
    timeout = 0,
    fast = None,
    columns = 1 )
```

Definition at line 459 of file [CameraAssembler.py](#).

```
00459     def readoff(self, waitOnSRAM=None, timeout=0, fast=None, columns=1):
00460         return self.comms.readoff(waitOnSRAM, timeout, fast, columns)
00461
```

6.1.3.48 readSerial()

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

Definition at line 465 of file [CameraAssembler.py](#).

```
00465     def readSerial(self, size, timeout=None):
00466         return self.comms.readSerial(size, timeout)
00467
```

6.1.3.49 readSRAM()

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

Definition at line 350 of file [CameraAssembler.py](#).

```
00350     def readSRAM(self):
00351         return self.board.readSRAM()
00352
```

6.1.3.50 reboot()

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

Perform soft reboot on board and reinitialize

Definition at line 636 of file [CameraAssembler.py](#).

```
00636     def reboot(self):
00637         """
00638         Perform soft reboot on board and reinitialize
00639         """
00640         self.board.softReboot()
00641         self.reinitialize()
00642
```

6.1.3.51 reinitialize()

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

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

Definition at line 622 of file [CameraAssembler.py](#).

```
00622     def reinitialize(self):
00623         """
00624         Reinitialize board registers and pots, reinitialize sensor timing (if
00625         previously set)
00626         """
00627         logging.info(self.loginfo + "reinitializing")
00628         self.initialize()
00629
00630         for side in self.senstiming:
00631             self.setTiming(side, self.senstiming[side][0], self.senstiming[side][1])
00632
00633         if self.sensmanual: # should be mutually exclusive with anything in senstiming
00634             self.setManualShutters(self.sensmanual)
00635
```

6.1.3.52 reportEdgeDetects()

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

Definition at line 392 of file [CameraAssembler.py](#).

```
00392     def reportEdgeDetects(self):
00393         return self.board.reportEdgeDetects()
00394
```

6.1.3.53 reportStatus()

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

Definition at line 389 of file [CameraAssembler.py](#).

```
00389     def reportStatus(self):
00390         return self.board.reportStatus()
00391
```

6.1.3.54 resetTimer()

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

Definition at line 359 of file [CameraAssembler.py](#).

```
00359     def resetTimer(self):
00360         return self.board.resetTimer()
00361
```

6.1.3.55 resolveSubreg()

```
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 801 of file [CameraAssembler.py](#).

```
00801 def resolveSubreg(self, srname):
00802     """
00803     Resolves subregister name or alias, returns object associated with subregister
00804     and flag indicating writability
00805
00806     Args:
00807         srname: name or alias of subregister
00808
00809     Returns:
00810         tuple(subregister name string, associated object, writable flag)
00811     """
00812     logging.debug(
00813         "{logdebug}resolveSubreg: srname = {srname}".format(
00814             logdebug=self.logdebug,
00815             srname=srname,
00816         )
00817     )
00818     writable = False
00819     srname = srname.upper()
00820     if srname in self.board.subreg_aliases:
00821         srname = self.board.subreg_aliases[srname].upper()
00822     if srname in self.board.subreglist:
00823         srobj = getattr(self.board, srname)
00824         writable = getattr(self.board, srname).writable
00825     else:
00826         # No-object error is handled by calling function
00827         srobj = None
00828     logging.debug(
00829         "{logdebug}resolveSubreg: srobj = {srobj}, writable={writable}".format(
00830             logdebug=self.logdebug, srobj=srobj, writable=writable
00831         )
00832     )
00833     return srname, srobj, writable
00834
00835
```

6.1.3.56 saveFrames()

```
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, saves it directly to disk for postprocessing. 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 1465 of file [CameraAssembler.py](#).

```
01465 def saveFrames(self, frames, path=None, filename="frames", prefix=None):
01466     """
01467     Save list of numpy arrays to disk. If passed an unprocessed text string, saves
01468     it directly to disk for postprocessing. Use 'prefix=""' for no prefix
01469
01470     Args:
01471     frames: numpy array or list of numpy arrays OR text string
01472     path: save path, defaults to './output'
01473     filename: defaults to 'frames.bin'
01474     prefix: prepended to filename, defaults to time/date (e.g. '160830-124704_')
01475
01476     Returns:
01477     Error string
01478     """
01479     logging.debug(
01480         self.logdebug
01481         + "saveFrames: path = "
01482         + str(path)
01483         + "; filename = "
01484         + str(filename)
01485         + "; prefix = "
01486         + str(prefix)
01487     )
01488     logging.info(self.loginfo + "saveFrames")
01489     err = ""
01490     if path is None:
01491         path = os.path.join(os.getcwd(), "output")
01492     if prefix is None:
01493         prefix = datetime.now().strftime("%y%m%d-%H%M%S%f")[:-5] + "_"
01494     if not os.path.exists(path):
01495         os.makedirs(path)
01496
01497     # TODO catch save file exceptions
01498     if isinstance(frames[0], str):
01499         logging.debug(self.logdebug + "saveFrames: saving text frames")
01500         filename = filename + ".txt"
01501         savefile = open(os.path.join(path, prefix + filename), "w+")
01502         savefile.write(frames)
01503     else:
01504         logging.debug(self.logdebug + "saveFrames: saving numerical frames")
01505         filename = filename + ".bin"
01506         stacked = np.stack(frames)
01507         try:
01508             stacked = stacked.reshape(
01509                 (
01510                     self.sensor.nframes,
01511                     self.sensor.height // (self.sensor.interlacing + 1),
01512                     self.sensor.width,
01513                 )
01514             )
01515         except Exception as e:
01516             err = self.logerr + "saveFrames: unable to save frames: " + str(e)
01517             logging.error(err)
01518
01519         stacked.tofile(os.path.join(path, prefix + filename))
01520     return err
01521
```

6.1.3.57 saveHDF()

```
nsCamera.CameraAssembler.CameraAssembler.saveHDF (
    self,
```

```

        frames,
        path = None,
        filename = "Acquisition",
        prefix = None )

```

Definition at line 2092 of file [CameraAssembler.py](#).

```

02098     ):
02099         """ """
02100         logging.info(self.loginfo + ": saveHDF")
02101         err = ""
02102         if path is None:
02103             path = os.path.join(os.getcwd(), "output")
02104         if prefix is None:
02105             prefix = datetime.now().strftime("%y%m%d-%H%M%S%f")[:-5] + "_"
02106         if not os.path.exists(path):
02107             os.makedirs(path)
02108
02109         h5file = os.path.join(path, prefix + filename + ".hdf5")
02110         with h5py.File(h5file, "w") as f:
02111             # shotgrp = f.create_group("DATA/SHOT")
02112             frame_index = 0
02113             for frame in frames:
02114                 grp = f.create_group("DATA/SHOT/FRAME_0" + str(frame_index))
02115                 data = grp.create_dataset(
02116                     "DATA", (self.sensor.height, self.sensor.width), data=frame
02117                 )
02118                 frame_index += 1
02119
02120
02121         """
02122         Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
02123         LLNL-CODE-838080
02124
02125         This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
02126         contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
02127         and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
02128         'nsCamera' is distributed under the terms of the MIT license. All new contributions must
02129         be made under this license.
02130         """

```

6.1.3.58 saveNumpys()

```

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 1522 of file [CameraAssembler.py](#).

```

01529     ):
01530         """
01531         Save numpy array or list of numpy arrays to disk as individual numpy data files,
01532         with frame number appended to filename.
01533
01534         Args:
01535             frames: numpy array or list of numpy arrays or single numpy array
01536             path: save path, defaults to './output'
01537             filename: defaults to 'Frame' followed by frame number
01538             prefix: prepended to 'filename', defaults to time/date
01539                   (e.g. '160830-124704_')
01540             index: number to start frame numbering
01541
01542         Returns:
01543             Error string
01544         """
01545         logging.info(self.logininfo + "saveNumpys")
01546         logging.debug(
01547             self.logdebug
01548             + "saveNumpys: path = "
01549             + str(path)
01550             + "; filename = "
01551             + str(filename)
01552             + "; prefix = "
01553             + str(prefix)
01554             + "; index = "
01555             + str(index)
01556         )
01557         err = ""
01558         if path is None:
01559             path = os.path.join(os.getcwd(), "output")
01560         if prefix is None:
01561             prefix = datetime.now().strftime("%y%m%d-%H%M%S%f")[:-5] + "_"
01562         if not os.path.exists(path):
01563             os.makedirs(path)
01564         if index is None:
01565             firstnum = self.sensor.firstframe
01566         else:
01567             firstnum = index
01568         if not isinstance(frames, list):
01569             frames = [frames]
01570
01571         # if this is a text string from fast readoff, do the numpy conversion now
01572         if isinstance(frames[0], str):
01573             frames = generateFrames(frames)
01574
01575         framestemp = np.copy(frames)
01576
01577         for idx, frame in enumerate(framestemp):
01578             if idx < len(framestemp) / 2:
01579                 interlacing = self.sensor.interlacing[0]
01580             else:
01581                 interlacing = self.sensor.interlacing[1]
01582             try:
01583                 if self.padToFull:
01584                     frame = np.reshape(
01585                         frame, (self.sensor.maxheight // (interlacing + 1), -1)
01586                     )
01587                 else:
01588                     frame = np.reshape(
01589                         frame,
01590                         (
01591                             (self.sensor.lastrow - self.sensor.firstrow + 1)
01592                             // (interlacing + 1),
01593                             -1,
01594                         ),
01595                     )
01596                 namenum = filename + "_%d" % firstnum
01597                 nppath = os.path.join(path, prefix + namenum + ".npy")
01598                 np.save(nppath, frame)
01599                 firstnum += 1
01600             except SystemExit:
01601                 raise
01602             except KeyboardInterrupt:
01603                 raise
01604             except Exception:
01605                 err = self.logerr + "saveNumpys: unable to save arrays"
01606                 logging.error(err)
01607             continue
01608         return err

```

01609

6.1.3.59 saveTiffs()

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

Definition at line 471 of file [CameraAssembler.py](#).

```
00471     def saveTiffs(self, frames, path=None, filename="Frame", prefix=None, index=None):
00472         return saveTiffs(self, frames, path, filename, prefix, index)
00473
```

6.1.3.60 selectOscillator()

```
nsCamera.CameraAssembler.CameraAssembler.selectOscillator (
    self,
    osc = None )
```

Definition at line 425 of file [CameraAssembler.py](#).

```
00425     def selectOscillator(self, osc=None):
00426         return self.sensor.selectOscillator(osc)
00427
```

6.1.3.61 sendCMD()

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

Definition at line 449 of file [CameraAssembler.py](#).

```
00449     def sendCMD(self, pkt):
00450         return self.comms.sendCMD(pkt)
00451
```

6.1.3.62 sensorSpecific()

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

Definition at line 422 of file [CameraAssembler.py](#).

```
00422     def sensorSpecific(self):
00423         return self.sensor.sensorSpecific()
00424
```

6.1.3.63 setArbTiming()

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

Definition at line 404 of file [CameraAssembler.py](#).

```
00404     def setArbTiming(self, side="AB", sequence=None):
00405         return self.sensor.setArbTiming(side, sequence)
00406
```

6.1.3.64 setExtClk()

```
nsCamera.CameraAssembler.CameraAssembler.setExtClk (
    self,
    dilation = None,
    frequency = None )
```

Definition at line 443 of file [CameraAssembler.py](#).

```
00443     def setExtClk(self, dilation=None, frequency=None):
00444         return self.sensor.setExtClk(dilation, frequency)
00445
```

6.1.3.65 setFrames()

```
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 1842 of file [CameraAssembler.py](#).

```
01842     def setFrames(self, minframe=None, maxframe=None):
01843         """
01844         Sets bounds on frames returned by board, inclusive (e.g., 0,3 returns four
01845         frames). If called without parameters, resets to full set of frames.
01846
01847         Args:
01848             minframe: first frame to read from board
01849             maxframe: last frame to read from board
01850
01851         Returns:
01852             Error string
01853         """
01854         logging.debug(
01855             self.logdebug
01856             + "setFrames: minframe = "
01857             + str(minframe)
```

```

01858         + "; maxframe = "
01859         + str(maxframe)
01860     )
01861     if minframe is None:
01862         minframe = self.sensor.minframe
01863     if maxframe is None:
01864         maxframe = self.sensor.maxframe
01865     if (
01866         not isinstance(minframe, int)
01867         or minframe < self.sensor.minframe
01868         or minframe > maxframe
01869         or not isinstance(maxframe, int)
01870         or maxframe > self.sensor.maxframe
01871     ):
01872         err = (
01873             self.logerr + "setFrames: invalid frame limits submitted. Frame "
01874             "selection remains unchanged. "
01875         )
01876         logging.error(err)
01877         return err
01878
01879     initframe = hex(minframe)[2:].zfill(8)
01880     finframe = hex(maxframe)[2:].zfill(8)
01881     err1, _ = self.setRegister("FPA_FRAME_INITIAL", initframe)
01882     err2, _ = self.setRegister("FPA_FRAME_FINAL", finframe)
01883     self.sensor.firstframe = minframe
01884     self.sensor.lastframe = maxframe
01885     self.sensor.nframes = maxframe - minframe + 1
01886     self.comms.payloadsize = (
01887         self.sensor.width
01888         * self.sensor.height
01889         * self.sensor.nframes
01890         * self.sensor.bytesperpixel
01891     )
01892     plural = ""
01893     if self.sensor.nframes > 1:
01894         plural = "s"
01895     logging.info(
01896         self.loginf
01897         + "Readoff set to "
01898         + str(self.sensor.nframes)
01899         + " frame"
01900         + plural
01901         + " ("
01902         + str(minframe)
01903         + ", "
01904         + str(maxframe)
01905         + ") "
01906     )
01907     err = err1 + err2
01908     if err:
01909         logging.error(
01910             self.logerr + "setFrames may not have functioned properly: " + err
01911         )
01912     return err
01913

```

6.1.3.66 setHighFullWell()

```

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

```

Definition at line 431 of file [CameraAssembler.py](#).

```

00431     def setHighFullWell(self, flag=True):
00432         return self.sensor.setHighFullWell(flag)
00433

```

6.1.3.67 setInterlacing()

```

nsCamera.CameraAssembler.CameraAssembler.setInterlacing (
    self,

```

```
        ifactor = None,  
        side = None )
```

Definition at line 428 of file [CameraAssembler.py](#).

```
00428     def setInterlacing(self, ifactor=None, side=None):  
00429         return self.sensor.setInterlacing(ifactor, side)  
00430
```

6.1.3.68 setLED()

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

Definition at line 365 of file [CameraAssembler.py](#).

```
00365     def setLED(self, LED=1, status=1):  
00366         return self.board.setLED(LED, status)  
00367
```

6.1.3.69 setManualShutters()

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

Definition at line 410 of file [CameraAssembler.py](#).

```
00410     def setManualShutters(self, timing=None):  
00411         return self.sensor.setManualTiming(timing)  
00412
```

6.1.3.70 setManualTiming()

```
nsCamera.CameraAssembler.CameraAssembler.setManualTiming (  
    self,  
    timing = None )
```

Definition at line 413 of file [CameraAssembler.py](#).

```
00413     def setManualTiming(self, timing=None):  
00414         return self.sensor.setManualTiming(timing)  
00415
```

6.1.3.71 setPhiDelay()

```
nsCamera.CameraAssembler.CameraAssembler.setPhiDelay (  
    self,  
    side = None,  
    delay = 0 )
```

Definition at line 440 of file [CameraAssembler.py](#).

```
00440     def setPhiDelay(self, side=None, delay=0):  
00441         return self.sensor.setPhiDelay(side, delay)  
00442
```

6.1.3.72 setPot()

```
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 1065 of file [CameraAssembler.py](#).

```
01065     def setPot(self, potname, value=1.0, errflag=False):
01066         """
01067         Sets value of pot to value, normalized so that '1.0' corresponds with the
01068         fixed point maximum value of pot.
01069
01070         Args:
01071             potname: common name of pot, e.g., VRST found in board.subreg_aliases or
01072             defined in board.subregisters
01073             value: float between 0 and 1
01074             errflag: if True, return tuple with error string
01075
01076         Returns:
01077             if errflag:
01078                 tuple: (error string, response packet as string)
01079             else:
01080                 response packet as string
01081         """
01082         logging.debug(
01083             "{logdebug}setPot: potname = {potname}; value={value} errflag = {errflag}"
01084             "".format(
01085                 logdebug=self.logdebug, potname=potname, value=value, errflag=errflag
01086             )
01087         )
01088
01089         if value < 0:
01090             value = 0.0
01091         if value > 1:
01092             value = 1.0
01093
01094         potname, potobj, writable = self.resolveSubreg(potname)
01095         if not potobj:
01096             err = "{logerr}setPot: invalid lookup: {potname}; returning '0'".format(
01097                 logerr=self.logerr, potname=potname
01098             )
01099
01100             logging.error(err)
01101             if errflag:
01102                 return err, 0
01103             return 0
01104         if not writable:
01105             err = "{logerr}setPot: not a writable subregister: {potname}; returning '0'"
01106             "".format(logerr=self.logerr, potname=potname)
01107             logging.error(err)
01108             if errflag:
01109                 return err, "0"
01110             return "0"
01111         setpoint = int(round(value * potobj.max_value))
```

```

01112         setpointpadded = "{num:{fill}{width}b}".format(
01113             num=setpoint, fill="0", width=potobj.width
01114         )
01115         logging.debug(
01116             "{logdebug}setpot: setpointpadded = {setpointpadded}".format(
01117                 logdebug=self.logdebug, setpointpadded=setpointpadded
01118             )
01119         )
01120
01121         err, rval = self.setSubregister(potname, setpointpadded)
01122         if err:
01123             logging.error(
01124                 err="{logerr}setPot: unable to confirm setting of subregister:"
01125                     " {potname}".format(logerr=self.logerr, potname=potname)
01126             )
01127             ident = potname[3:]
01128             if ident[0].isdigit(): # numbered pot scheme
01129                 potnumlatch = int(ident) * 2 + 1
01130                 potnumlatchstring = "{num:{fill}{width}x}".format(
01131                     num=potnumlatch, fill="0", width=8
01132                 )
01133                 err1, resp = self.setRegister("POT_CTL", potnumlatchstring)
01134             else: # alphabetical DAC scheme
01135                 ident = ident.upper() # expects single character, e.g. 'A' from 'DACA'
01136                 identnum = ord(ident) - ord("A") # DACA -> 0
01137                 potnumlatch = int(identnum) * 2 + 1
01138                 potnumlatchstring = "{num:{fill}{width}x}".format(
01139                     num=potnumlatch, fill="0", width=8
01140                 )
01141                 err1, resp = self.setRegister("DAC_CTL", potnumlatchstring)
01142             if err1:
01143                 # logging.error(self.logerr + "setPot: unable to latch register")
01144
01145                 logging.error(
01146                     err="{logerr}setPot: unable to latch register".format(
01147                         logerr=self.logerr
01148                     )
01149                 )
01150
01151             if errflag:
01152                 return err + err1, rval
01153             return rval
01154

```

6.1.3.73 setPotV()

```

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 the 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 1197 of file [CameraAssembler.py](#).

```

01206     ):
01207         """
01208         Sets pot to specified voltage. If tune=True, uses monitor to adjust pot to
01209         correct voltage. Tuning will attempt to tune to closest LSB on pot; if
01210         'accuracy' > LSB resolution, will only complain if tuning is unable to get
01211         the voltage within 'accuracy'
01212
01213         Args:
01214             potname: common name of pot, e.g., VRST found in board.subreg_aliases or
01215                     defined in board.subregisters
01216             voltage: voltage bound by pot max and min (set in board object)
01217             tune: if True, iterate with monitor to correct voltage
01218             accuracy: acceptable error in volts (if None, attempts to find the closest
01219                     possible pot setting and warns if last iteration does not reduce error
01220                     below the resolution of the pot)
01221             iterations: number of iteration attempts
01222             approach: approximation parameter (>1 may cause overshoot)
01223             errflag: if True, return tuple with error string
01224
01225         Returns:
01226             if errflag:
01227                 tuple: (error string, response string)
01228             else:
01229                 response string
01230         """
01231         logging.debug(
01232             self.logdebug
01233             + "setPotV: potname = "
01234             + str(potname)
01235             + "; voltage = "
01236             + str(voltage)
01237             + "; tune = "
01238             + str(tune)
01239             + "; accuracy = "
01240             + str(accuracy)
01241             + "; iterations = "
01242             + str(iterations)
01243             + "; approach = "
01244             + str(approach)
01245             + "; errflag = "
01246             + str(errflag)
01247         )
01248         potname, potobj, writable = self.resolveSubreg(potname)
01249         if not potobj:
01250             err = (
01251                 self.logerr
01252                 + "setPotV: invalid lookup: "
01253                 + potname
01254                 + " , returning zero "
01255             )
01256             logging.error(err)
01257             if errflag:
01258                 return err, 0
01259             return 0
01260         if not writable:
01261             err = (
01262                 self.logerr
01263                 + "setPotV: not a writable subregister: "
01264                 + potname
01265                 + "; returning zero"
01266             )
01267             logging.error(err)
01268             if errflag:
01269                 return err, 0
01270             return 0

```



```

01271         if voltage < potobj.minV:
01272             voltage = potobj.minV
01273         if voltage > potobj.maxV:
01274             voltage = potobj.maxV
01275         setting = (voltage - potobj.minV) / (potobj.maxV - potobj.minV)
01276         logging.debug(self.logdebug + "setPotV: setting = " + str(setting))
01277         err, rval = self.setPot(potname, setting, errflag=True)
01278         time.sleep(0.1)
01279         # TODO: refactor tuning to separate method
01280         if tune:
01281             logging.debug(self.logdebug + "setPotV: beginning tuning")
01282             if potname not in self.board.monitor_controls.values():
01283                 err = (
01284                     self.logerr
01285                     + "setPotV: pot ' "
01286                     + potname
01287                     + "' does not have a corresponding monitor"
01288                 )
01289             logging.error(err)
01290             if errflag:
01291                 return err, rval
01292             return rval
01293         self.setPot(potname, 0.65)
01294         time.sleep(0.2)
01295         err1, mon65 = self.getMonV(potname, errflag=True)
01296         self.setPot(potname, 0.35)
01297         time.sleep(0.2)
01298         err2, mon35 = self.getMonV(potname, errflag=True)
01299         # theoretical voltage range assuming linearity
01300         potrange = (mon65 - mon35) / 0.3
01301         stepsize = potrange / (potobj.max_value + 1)
01302         err += err1 + err2
01303         if err or potrange < 1:
01304             err += self.logerr + "setPotV: unable to tune pot " + potname
01305             if potrange < 1: # potrange should be on the order of 3.3 or 5 volts
01306                 err += "; monitor shows insufficient change with pot variation; "
01307                 "retrying setPotV with tune=False"
01308             logging.warning(err)
01309             err, rval = self.setPotV(
01310                 potname=potname, voltage=voltage, tune=False, errflag=True
01311             )
01312             if errflag:
01313                 return err, rval
01314             return rval
01315         potzero = 0.35 - (mon35 / potrange)
01316         potone = 1.65 - (mon65 / potrange)
01317         if potzero < 0:
01318             potzero = 0
01319         if potone > 1:
01320             potone = 1
01321
01322         if accuracy > stepsize:
01323             mindiff = accuracy
01324         else:
01325             mindiff = stepsize
01326         setting = potzero + (voltage / potone)
01327         self.setPot(potname, setting)
01328         lastdiff = 0
01329         smalladjust = 0
01330         err3 = ""
01331         for _ in range(iterations):
01332             err3i, measured = self.getMonV(potname, errflag=True)
01333             if err3i:
01334                 err3 = err3 + err3i + " "
01335             diff = voltage - measured
01336             if abs(diff - lastdiff) < stepsize / 2:
01337                 if (
01338                     smalladjust > 12
01339                 ): # magic number for now; if it doesn't converge after several
01340                     # tries, it never will, usually because the setting is pinned
01341                     # to 0 or 1 and adjust can't change it
01342                     logging.warning(
01343                         self.logwarn
01344                         + "setPotV: Tuning converged too slowly: pot "
01345                         + potname
01346                         + " set to "
01347                         + str(voltage)
01348                         + "V, monitor returns "
01349                         + str(measured)
01350                         + "V; if this value is incorrect, consider trying "
01351                         + "tune=False"

```

```

01352         )
01353         logging.debug(self.logdebug + "setPotV: tuning complete")
01354         if errflag:
01355             return "", rval
01356         return rval
01357         smalladjust += 1
01358         if not int(2 * diff / stepsize):
01359             # TODO: is this check redundant with the first one?
01360             logging.debug(self.logdebug + "setPotV: tuning complete")
01361             if errflag:
01362                 return "", rval
01363             return rval
01364         adjust = approach * (diff / potrange)
01365         setting += adjust
01366         if setting > 1:
01367             setting = 1
01368         elif setting < 0:
01369             setting = 0
01370         err1, rval = self.setPot(potname, setting, True)
01371         lastdiff = diff
01372         time.sleep(0.2)
01373         err4, measured = self.getMonV(potname, errflag=True)
01374         diff = voltage - measured
01375         # code will try to get to within one stepsize, but will only complain if it
01376         # doesn't get within mindiff
01377         if int(diff / mindiff):
01378             logging.warning(
01379                 self.logwarn
01380                 + "setPotV: pot "
01381                 + potname
01382                 + " set to "
01383                 + str(voltage)
01384                 + "V, monitor returns "
01385                 + str(measured)
01386                 + "V"
01387             )
01388         err += err1 + err2 + err3 + err4
01389         if err:
01390             logging.error(self.logerr + "setPotV: errors occurred: " + err)
01391         if errflag:
01392             return err, rval
01393         logging.debug(self.logdebug + "setPotV: tuning complete")
01394         return rval
01395

```

6.1.3.74 setPowerSave()

```

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

```

Definition at line 368 of file [CameraAssembler.py](#).

```

00368     def setPowerSave(self, status=1):
00369         return self.board.setPowerSave(status)
00370

```

6.1.3.75 setPPER()

```

nsCamera.CameraAssembler.CameraAssembler.setPPER (
    self,
    pollperiod = None )

```

Definition at line 371 of file [CameraAssembler.py](#).

```

00371     def setPPER(self, pollperiod=None):
00372         return self.board.setPPER(pollperiod)
00373

```

6.1.3.76 setRegister()

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

Sets named register to given value

Args:

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

Returns:

tuple: (error string, response string)

Definition at line 743 of file [CameraAssembler.py](#).

```
00743     def setRegister(self, regname, regval):
00744         """
00745         Sets named register to given value
00746
00747         Args:
00748             regname: name of register as given in ICD
00749             regval: value to assign to register, as integer or hexadecimal string
00750                     with or without '0x'
00751
00752         Returns:
00753             tuple: (error string, response string)
00754         """
00755         logging.debug(
00756             "{logdebug}setRegister: regname = {regname}; regval = {regval}".format(
00757                 logdebug=self.logdebug, regname=regname, regval=regval
00758             )
00759         )
00760
00761         regname = regname.upper()
00762         if regname not in self.board.registers:
00763             err = "{logerr}setRegister: Invalid register name: {regname}".format(
00764                 logerr=self.logerr, regname=regname
00765             )
00766             logging.error(err)
00767             return err, "00000000"
00768         if isinstance(regval, int):
00769             regval = hex(regval)
00770         try:
00771             if regval[0:2] == "0x":
00772                 regval = regval[2:]
00773         except TypeError:
00774             err = "{logerr}setRegister: invalid register value parameter".format(
00775                 logerr=self.logerr
00776             )
00777             logging.error(err)
00778             return err, "00000000"
00779         pkt = Packet(addr=self.board.registers[regname], data=regval)
00780         err, rval = self.comms.sendCMD(pkt)
00781         if err:
00782             logging.error(
00783                 "{logerr}setRegister: {regname}: {err}".format(
00784                     logerr=self.logerr, regname=regname, err=err
00785                 )
00786             )
00787         if len(rval) < 32:
00788             logging.debug(
00789                 "{logdebug}SetRegister: rval = {rval}".format(
00790                     logdebug=self.logdebug, rval=rval
00791                 )
00792             )
00793         else:
00794             logging.debug(
00795                 "{logdebug}SetRegister: rval (truncated)= {rval}".format(
00796                     logdebug=self.logdebug, rval=rval[0:32]
00797                 )
00798             )
00799         return err, rval
00800
```

6.1.3.77 setRows()

```
nsCamera.CameraAssembler.CameraAssembler.setRows (
    self,
    minrow = 0,
    maxrow = None,
    padToFull = 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
padToFull: if True, generate full size frames, padding collected rows with
           zeroes if necessary
```

Definition at line 1914 of file [CameraAssembler.py](#).

```
01914     def setRows(self, minrow=0, maxrow=None, padToFull=False):
01915         """
01916         Sets bounds on rows returned by board, inclusive (e.g., 0,1023 returns all 1024
01917         rows). If called without parameters, resets to full image size.
01918
01919         Args:
01920             minrow: first row to return from board
01921             maxrow: last row to return from board
01922             padToFull: if True, generate full size frames, padding collected rows with
01923                       zeroes if necessary
01924         """
01925         logging.debug(
01926             self.logdebug
01927             + "setRows: minrow = "
01928             + str(minrow)
01929             + "; maxrow = "
01930             + str(maxrow)
01931             + "; padToFull = "
01932             + str(padToFull)
01933         )
01934         if maxrow is None:
01935             maxrow = self.sensor.maxheight - 1
01936         if (
01937             not isinstance(minrow, int)
01938             or minrow < 0
01939             or minrow > maxrow
01940             or not isinstance(maxrow, int)
01941             or maxrow >= self.sensor.maxheight
01942         ):
01943             err = (
01944                 self.logerr + "setRows: invalid row arguments submitted. Frame size"
01945                 + " remains unchanged. "
01946             )
01947             logging.error(err)
01948             return err
01949
01950         initrow = hex(minrow)[2:].zfill(8)
01951         finrow = hex(maxrow)[2:].zfill(8)
01952         err1, _ = self.setRegister("FPA_ROW_INITIAL", initrow)
01953         err2, _ = self.setRegister("FPA_ROW_FINAL", finrow)
01954         self.sensor.firstrow = minrow
01955         self.sensor.lastrow = maxrow
01956         self.sensor.height = maxrow - minrow + 1
01957         self.comms.payloadsize = (
01958             self.sensor.width
01959             * self.sensor.height
01960             * self.sensor.nframes
01961             * self.sensor.bytesperpixel
01962         )
01963
01964         if self.commname == "rs422":
01965             self.comms.datatimeout = (
01966                 (1.0 * self.sensor.height / self.sensor.maxheight)
01967                 * 5e7
```

```

01968             * self.sensor.nframes
01969             / self.comms.baud
01970         )
01971     self.padToFull = padToFull
01972     logging.info(
01973         self.loginfo
01974         + "Readoff set to "
01975         + str(self.sensor.height)
01976         + " rows ("
01977         + str(minrow)
01978         + ", "
01979         + str(maxrow)
01980         + ") "
01981     )
01982     err = err1 + err2
01983     if err:
01984         logging.error(
01985             self.logerr + "setRows may not have functioned properly: " + err
01986         )
01987     return err
01988 
```

6.1.3.78 setSubregister()

```

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: integer or binary string with or without '0b'

Returns:

tuple: (error, packet response string) from setRegister

Definition at line 883 of file [CameraAssembler.py](#).

```

00883     def setSubregister(self, subregname, valstring):
00884         """
00885         Sets substring of register identified in board attribute 'subregname' to
00886         valstring if subregister is writable
00887
00888         Args:
00889             subregname: listed in board.subreg_aliases or defined in board.subregisters
00890             valstring: integer or binary string with or without '0b'
00891
00892         Returns:
00893             tuple: (error, packet response string) from setRegister
00894         """
00895         logging.debug(
00896             "{logdebug}setSubregister: subregname = {subregname}; valstring ="
00897             " {valstring}".format(
00898                 logdebug=self.logdebug, subregname=subregname, valstring=valstring
00899             )
00900         )
00901
00902         subregname, subregobj, writable = self.resolveSubreg(subregname)
00903         if not subregobj:
00904             err = "{logerr}getSubregister: invalid lookup: {subregname}".format(
00905                 logerr=self.logerr, subregname=subregname
00906             )
00907
00908             logging.error(err)
00909             return err, "0"
00910         if not writable:
00911             err = "{logerr}getSubregister: not a writable subregister: {subregname}"

```

```

00912         "%.format(logerr=self.logerr, subregname=subregname)
00913         logging.error(err)
00914         return err, "0"
00915     if isinstance(valstring, int):
00916         valstring = bin(valstring)[2:]
00917     try:
00918         if valstring[0:2] == "0b":
00919             valstring = valstring[2:]
00920     except TypeError:
00921         err = "{logerr}getSubregister: invalid subregister value parameter".format(
00922             logerr=self.logerr
00923         )
00924         logging.error(err)
00925         return err, "0"
00926     if len(str(valstring)) > subregobj.width:
00927         err = "{logerr}getSubregister: ivalue string is too long".format(
00928             logerr=self.logerr
00929         )
00930         logging.error(err)
00931         return err, "0"
00932     # read current value of register data
00933     err, resp = self.getRegister(subregobj.register)
00934     if err:
00935         logging.error(
00936             "{logerr}getSubregister: unable to retrieve register setting; setting"
00937             " of {subregname} likely failed ".format(
00938                 logerr=self.logerr, subregname=subregname
00939             )
00940         )
00941         return err, "0"
00942     hex_str = "0x" + resp
00943     b_reg_value = "{0:0=32b}".format(int(hex_str, 16)) # convert to binary
00944     # list indexing is reversed from bit string; the last bit of the string is at
00945     # index 0 in the list (thus bit 0 is at index 0)
00946     startindex = 31 - subregobj.start_bit
00947     valstringpadded = str(valstring).zfill(subregobj.width)
00948     fullreg = list(b_reg_value)
00949     fullreg[startindex : startindex + subregobj.width] = valstringpadded
00950     # convert binary string back to hexadecimal string for writing
00951     new_reg_value = "".join(fullreg)
00952     h_reg_value = "{num:{fill}{width}x}".format(
00953         num=int(new_reg_value, 2), fill="0", width=8
00954     )
00955     err, retval = self.setRegister(subregobj.register, h_reg_value)
00956     # logging.debug(self.logdebug + "retval = " + str(retval))
00957     if len(retval) < 32:
00958         logging.debug(
00959             "{logdebug}setSubregister: retval = {retval}".format(
00960                 logdebug=self.logdebug, retval=retval
00961             )
00962         )
00963     else:
00964         logging.debug(
00965             "{logdebug}setSubregister: retval (truncated) = {retval}".format(
00966                 logdebug=self.logdebug, retval=retval[0:32]
00967             )
00968         )
00969     return err, retval
00970
00971
00972
00973
00974

```

6.1.3.79 setTiming()

```

nsCamera.CameraAssembler.CameraAssembler.setTiming (
    self,
    side = "AB",
    sequence = None,
    delay = 0 )

```

Definition at line 401 of file [CameraAssembler.py](#).

```

00401     def setTiming(self, side="AB", sequence=None, delay=0):
00402         return self.sensor.setTiming(side, sequence, delay)
00403

```

6.1.3.80 setTriggerDelay()

```
nsCamera.CameraAssembler.CameraAssembler.setTriggerDelay (
    self,
    delay = 0 )
```

Definition at line 437 of file [CameraAssembler.py](#).

```
00437     def setTriggerDelay(self, delay=0):
00438         return self.sensor.setTriggerDelay(delay)
00439
```

6.1.3.81 setZeroDeadTime()

```
nsCamera.CameraAssembler.CameraAssembler.setZeroDeadTime (
    self,
    flag = True,
    side = None )
```

Definition at line 434 of file [CameraAssembler.py](#).

```
00434     def setZeroDeadTime(self, flag=True, side=None):
00435         return self.sensor.setZeroDeadTime(flag, side)
00436
```

6.1.3.82 startCapture()

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

Definition at line 347 of file [CameraAssembler.py](#).

```
00347     def startCapture(self, mode):
00348         return self.board.startCapture(mode)
00349
```

6.1.3.83 str2bytes()

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

Definition at line 483 of file [CameraAssembler.py](#).

```
00483     def str2bytes(self, astring):
00484         return str2bytes(astring)
00485
```

6.1.3.84 str2nparray()

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

Definition at line 489 of file [CameraAssembler.py](#).

```
00489     def str2nparray(self, valstring):
00490         return str2nparray(valstring)
00491
```

6.1.3.85 submitMessages()

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

Serially set multiple register / subregister values

Args:

messages: list of tuples (register name, integer or hexadecimal string with or without '0x') and/or (subregister name, integer or binary string with or without '0b')

errorstring: error message to print in case of failure

Returns:

tuple (accumulated error string, response string of final message)

Definition at line 975 of file [CameraAssembler.py](#).

```
00975     def submitMessages(self, messages, errorstring="Error"):
00976         """
00977         Serially set multiple register / subregister values
00978
00979         Args:
00980             messages: list of tuples (register name, integer or hexadecimal string with
00981                 or without '0x') and/or (subregister name, integer or binary string with
00982                 or without '0b')
00983             errorstring: error message to print in case of failure
00984
00985         Returns:
00986             tuple (accumulated error string, response string of final message)
00987         """
00988         logging.debug(
00989             "{logdebug}submitMessages: messages = {messages}; errorstring ="
00990             " {errorstring}".format(
00991                 logdebug=self.logdebug, messages=messages, errorstring=errorstring
00992             )
00993         )
00994
00995         errs = ""
00996         err = ""
00997         rval = ""
00998         for m in messages:
00999             if m[0].upper() in self.board.registers:
01000                 err, rval = self.setRegister(m[0].upper(), m[1])
01001             elif m[0].upper() in self.board.subreglist:
01002                 err, rval = self.setSubregister(m[0].upper(), m[1])
01003             else:
01004                 err = "{logerr}submitMessages: Invalid register/subregister:"
01005                 " {errorstring}:{m0}; ".format(
01006                     logerr=self.logerr, errorstring=errorstring, m0=m[0]
01007                 )
01008
01009                 logging.error(err)
01010                 errs = errs + err
01011         return err, rval
01012
```

6.1.3.86 waitForSRAM()

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

Definition at line 353 of file [CameraAssembler.py](#).

```
00353     def waitForSRAM(self, timeout=None):
00354         return self.board.waitForSRAM(timeout)
00355
```


6.1.3.87 writeSerial()

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

Definition at line 462 of file [CameraAssembler.py](#).

```
00462     def writeSerial(self, cmd, timeout=None):
00463         return self.comms.writeSerial(cmd, timeout)
00464
```

6.1.4 Member Data Documentation

6.1.4.1 abort

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

Definition at line 252 of file [CameraAssembler.py](#).

6.1.4.2 arch

```
nsCamera.CameraAssembler.CameraAssembler.arch
```

Definition at line 228 of file [CameraAssembler.py](#).

6.1.4.3 armed

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

Definition at line 244 of file [CameraAssembler.py](#).

6.1.4.4 board

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

Definition at line 549 of file [CameraAssembler.py](#).

6.1.4.5 boardname

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

Definition at line 204 of file [CameraAssembler.py](#).

6.1.4.6 commname

`nsCamera.CameraAssembler.CameraAssembler.commname`

Definition at line 211 of file [CameraAssembler.py](#).

6.1.4.7 comms

`nsCamera.CameraAssembler.CameraAssembler.comms`

Definition at line 543 of file [CameraAssembler.py](#).

6.1.4.8 currtime

`nsCamera.CameraAssembler.CameraAssembler.currtime`

Definition at line 195 of file [CameraAssembler.py](#).

6.1.4.9 cycle

`nsCamera.CameraAssembler.CameraAssembler.cycle`

Definition at line 203 of file [CameraAssembler.py](#).

6.1.4.10 FPGAboardtype

`nsCamera.CameraAssembler.CameraAssembler.FPGAboardtype`

Definition at line 234 of file [CameraAssembler.py](#).

6.1.4.11 FPGAinterfaces

`nsCamera.CameraAssembler.CameraAssembler.FPGAinterfaces`

Definition at line 237 of file [CameraAssembler.py](#).

6.1.4.12 FPGAinvalid

`nsCamera.CameraAssembler.CameraAssembler.FPGAinvalid`

Definition at line 240 of file [CameraAssembler.py](#).

6.1.4.13 FPGANum

`nsCamera.CameraAssembler.CameraAssembler.FPGANum`

Definition at line 231 of file [CameraAssembler.py](#).

6.1.4.14 FPGArad

`nsCamera.CameraAssembler.CameraAssembler.FPGArad`

Definition at line 235 of file [CameraAssembler.py](#).

6.1.4.15 FPGAsensor

`nsCamera.CameraAssembler.CameraAssembler.FPGAsensor`

Definition at line 236 of file [CameraAssembler.py](#).

6.1.4.16 FPGAVersion

`nsCamera.CameraAssembler.CameraAssembler.FPGAVersion`

Definition at line 230 of file [CameraAssembler.py](#).

6.1.4.17 inittime

`nsCamera.CameraAssembler.CameraAssembler.inittime`

Definition at line 250 of file [CameraAssembler.py](#).

6.1.4.18 iplist

`nsCamera.CameraAssembler.CameraAssembler.iplist`

Definition at line 242 of file [CameraAssembler.py](#).

6.1.4.19 logcrit

`nsCamera.CameraAssembler.CameraAssembler.logcrit`

Definition at line 272 of file [CameraAssembler.py](#).

6.1.4.20 logcritbase

`nsCamera.CameraAssembler.CameraAssembler.logcritbase`

Definition at line 266 of file [CameraAssembler.py](#).

6.1.4.21 logdebug

`nsCamera.CameraAssembler.CameraAssembler.logdebug`

Definition at line 276 of file [CameraAssembler.py](#).

6.1.4.22 logdebugbase

`nsCamera.CameraAssembler.CameraAssembler.logdebugbase`

Definition at line 270 of file [CameraAssembler.py](#).

6.1.4.23 logerr

`nsCamera.CameraAssembler.CameraAssembler.logerr`

Definition at line 273 of file [CameraAssembler.py](#).

6.1.4.24 logerrbase

`nsCamera.CameraAssembler.CameraAssembler.logerrbase`

Definition at line 267 of file [CameraAssembler.py](#).

6.1.4.25 loginfo

`nsCamera.CameraAssembler.CameraAssembler.loginfo`

Definition at line 275 of file [CameraAssembler.py](#).

6.1.4.26 loginfobase

`nsCamera.CameraAssembler.CameraAssembler.loginfobase`

Definition at line 269 of file [CameraAssembler.py](#).

6.1.4.27 logtag

`nsCamera.CameraAssembler.CameraAssembler.logtag`

Definition at line 264 of file [CameraAssembler.py](#).

6.1.4.28 logwarn

`nsCamera.CameraAssembler.CameraAssembler.logwarn`

Definition at line 274 of file [CameraAssembler.py](#).

6.1.4.29 logwarnbase

`nsCamera.CameraAssembler.CameraAssembler.logwarnbase`

Definition at line 268 of file [CameraAssembler.py](#).

6.1.4.30 oldtime

`nsCamera.CameraAssembler.CameraAssembler.oldtime`

Definition at line 196 of file [CameraAssembler.py](#).

6.1.4.31 packageroot

`nsCamera.CameraAssembler.CameraAssembler.packageroot`

Definition at line 243 of file [CameraAssembler.py](#).

6.1.4.32 padToFull

`nsCamera.CameraAssembler.CameraAssembler.padToFull`

Definition at line 251 of file [CameraAssembler.py](#).

6.1.4.33 parsedtime

`nsCamera.CameraAssembler.CameraAssembler.parsedtime`

Definition at line 201 of file [CameraAssembler.py](#).

6.1.4.34 `payloaderror`

`nsCamera.CameraAssembler.CameraAssembler.payloaderror`

Definition at line 320 of file [CameraAssembler.py](#).

6.1.4.35 `platform`

`nsCamera.CameraAssembler.CameraAssembler.platform`

Definition at line 227 of file [CameraAssembler.py](#).

6.1.4.36 `port`

`nsCamera.CameraAssembler.CameraAssembler.port`

Definition at line 224 of file [CameraAssembler.py](#).

6.1.4.37 `PY3`

`nsCamera.CameraAssembler.CameraAssembler.PY3`

Definition at line 226 of file [CameraAssembler.py](#).

6.1.4.38 `pyth1`

`nsCamera.CameraAssembler.CameraAssembler.pyth1`

Definition at line 225 of file [CameraAssembler.py](#).

6.1.4.39 `pyth2`

`nsCamera.CameraAssembler.CameraAssembler.pyth2`

Definition at line 225 of file [CameraAssembler.py](#).

6.1.4.40 `python`

`nsCamera.CameraAssembler.CameraAssembler.python`

Definition at line 225 of file [CameraAssembler.py](#).

6.1.4.41 read

`nsCamera.CameraAssembler.CameraAssembler.read`

Definition at line 199 of file [CameraAssembler.py](#).

6.1.4.42 savetime

`nsCamera.CameraAssembler.CameraAssembler.savetime`

Definition at line 202 of file [CameraAssembler.py](#).

6.1.4.43 sensmanual

`nsCamera.CameraAssembler.CameraAssembler.sensmanual`

Definition at line 249 of file [CameraAssembler.py](#).

6.1.4.44 sensor

`nsCamera.CameraAssembler.CameraAssembler.sensor`

Definition at line 524 of file [CameraAssembler.py](#).

6.1.4.45 sensorname

`nsCamera.CameraAssembler.CameraAssembler.sensorname`

For regular version.

Definition at line 216 of file [CameraAssembler.py](#).

6.1.4.46 senstiming

`nsCamera.CameraAssembler.CameraAssembler.senstiming`

Definition at line 248 of file [CameraAssembler.py](#).

6.1.4.47 timeout

`nsCamera.CameraAssembler.CameraAssembler.timeout`

Definition at line 205 of file [CameraAssembler.py](#).

6.1.4.48 `trigtime`

`nsCamera.CameraAssembler.CameraAssembler.trigtime`

Definition at line 197 of file [CameraAssembler.py](#).

6.1.4.49 `unstringed`

`nsCamera.CameraAssembler.CameraAssembler.unstringed`

Definition at line 200 of file [CameraAssembler.py](#).

6.1.4.50 `verblevel`

`nsCamera.CameraAssembler.CameraAssembler.verblevel`

Definition at line 278 of file [CameraAssembler.py](#).

6.1.4.51 `verbmap`

`nsCamera.CameraAssembler.CameraAssembler.verbmap`

Definition at line 254 of file [CameraAssembler.py](#).

6.1.4.52 `verbose`

`nsCamera.CameraAssembler.CameraAssembler.verbose`

Definition at line 223 of file [CameraAssembler.py](#).

6.1.4.53 `version`

`nsCamera.CameraAssembler.CameraAssembler.version`

Definition at line 194 of file [CameraAssembler.py](#).

6.1.4.54 `waited`

`nsCamera.CameraAssembler.CameraAssembler.waited`

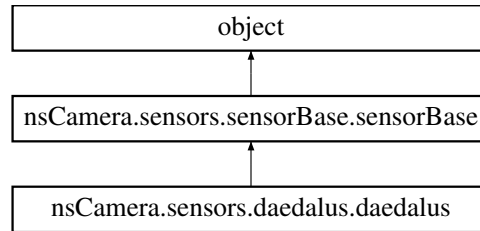
Definition at line 198 of file [CameraAssembler.py](#).

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

- [C:/Users/hill35/git/camera_python/nsCamera/CameraAssembler.py](#)

6.2 nsCamera.sensors.daedalus.daedalus Class Reference

Inheritance diagram for nsCamera.sensors.daedalus.daedalus:



Public Member Functions

- `__init__` (self, [ca](#))
- `sensorSpecific` (self)
- `setInterlacing` (self, ifactor=None, side=None)
- `setHighFullWell` (self, flag)
- `setZeroDeadTime` (self, flag=True, side=None)
- `selectOscillator` (self, osc=None)
- `setTriggerDelay` (self, delay=0)
- `setPhiDelay` (self, side=None, delay=0)
- `setExtClk` (self, dilation=None, frequency=None)
- `setManualShutters` (self, timing=None)
- `setManualTiming` (self, timing=None)
- `getManualTiming` (self)
- `getSensTemp` (self, scale=None, offset=None, slope=None, dec=1)
- `parseReadoff` (self, frames, [columns](#))
- `reportStatusSensor` (self, statusbits, statusbits2)

Public Member Functions inherited from [nsCamera.sensors.sensorBase.sensorBase](#)

- `init_board_specific` (self)
- `checkSensorVoltStat` (self)
- `setTiming` (self, side="AB", sequence=None, delay=0)
- `setArbTiming` (self, side="AB", sequence=None)
- `getTiming` (self, side, actual)
- `getSensorStatus` (self)

Public Attributes

- [ca](#)
- [sens_registers](#)
- [sens_subregisters](#)
- [interlacing](#)
- [columns](#)
- [HFW](#)
- [ZDT](#)

Public Attributes inherited from [nsCamera.sensors.sensorBase.sensorBase](#)

- [ca](#)
- [logcrit](#)
- [logerr](#)
- [logwarn](#)
- [loginfo](#)
- [logdebug](#)
- [fpganumID](#)
- [sensfam](#)

Static Public Attributes

- str [specwarn](#) = ""
- int [minframe](#) = 0
- int [maxframe](#) = 2
- int [maxwidth](#) = 512
- int [maxheight](#) = 1024
- int [bytesperpixel](#) = 2
- int [fpganumID](#) = 2
- str [detect](#) = "DAEDALUS_DET"
- str [sensfam](#) = "Daedalus"
- str [loglabel](#) = "[Daedalus] "
- bool [ZDT](#) = False
- bool [HFW](#) = False
- int [firstframe](#) = 0
- int [lastframe](#) = 2
- int [nframes](#) = 3
- int [width](#) = 512
- int [height](#) = 1024
- int [firstrow](#) = 0
- int [lastrow](#) = 1023
- list [interlacing](#) = [0, 0]
- int [columns](#) = 1
- bool [padToFull](#) = True
- float [toffset](#) = -165.76
- float [tslope](#) = 81.36

6.2.1 Detailed Description

Definition at line 30 of file [daedalus.py](#).

6.2.2 Constructor & Destructor Documentation

6.2.2.1 __init__()

```
nsCamera.sensors.daedalus.daedalus.__init__ (
    self,
    ca )
```

Reimplemented from [nsCamera.sensors.sensorBase.sensorBase](#).

Definition at line 56 of file [daedalus.py](#).

```
00056     def __init__(self, ca):
00057         self.ca = ca
00058         super(daedalus, self).__init__(ca)
00059
00060         self.sens_registers = OrderedDict(
00061             {
00062                 "HST_READBACK_A_LO": "018",
00063                 "HST_READBACK_A_HI": "019",
00064                 "HST_READBACK_B_LO": "01A",
00065                 "HST_READBACK_B_HI": "01B",
00066                 "HSTALLWEN_WAIT_TIME": "03F",
00067                 "VRESET_HIGH_VALUE": "04A",
00068                 "FRAME_ORDER_SEL": "04B",
00069                 "EXT_PHI_CLK_SH0_ON": "050",
00070                 "EXT_PHI_CLK_SH0_OFF": "051",
00071                 "EXT_PHI_CLK_SH1_ON": "052",
00072                 "EXT_PHI_CLK_SH1_OFF": "053",
00073                 "EXT_PHI_CLK_SH2_ON": "054",
00074                 "HST_TRIGGER_DELAY_DATA_LO": "120",
00075                 "HST_TRIGGER_DELAY_DATA_HI": "121",
00076                 "HST_PHI_DELAY_DATA": "122",
00077                 "HST_EXT_CLK_HALF_PER": "129",
00078                 "HST_COUNT_TRIG": "130",
00079                 "HST_DELAY_EN": "131",
00080                 "RSL_HFW_MODE_EN": "133",
00081                 "RSL_ZDT_MODE_B_EN": "135",
00082                 "RSL_ZDT_MODE_A_EN": "136",
00083                 "BGTRIMA": "137",
00084                 "BGTRIMB": "138",
00085                 "COLUMN_TEST_EN": "139",
00086                 "RSL_CONFIG_DATA_B0": "140",
00087                 "RSL_CONFIG_DATA_B1": "141",
00088                 "RSL_CONFIG_DATA_B2": "142",
00089                 "RSL_CONFIG_DATA_B3": "143",
00090                 "RSL_CONFIG_DATA_B4": "144",
00091                 "RSL_CONFIG_DATA_B5": "145",
00092                 "RSL_CONFIG_DATA_B6": "146",
00093                 "RSL_CONFIG_DATA_B7": "147",
00094                 "RSL_CONFIG_DATA_B8": "148",
00095                 "RSL_CONFIG_DATA_B9": "149",
00096                 "RSL_CONFIG_DATA_B10": "14A",
00097                 "RSL_CONFIG_DATA_B11": "14B",
00098                 "RSL_CONFIG_DATA_B12": "14C",
00099                 "RSL_CONFIG_DATA_B13": "14D",
00100                 "RSL_CONFIG_DATA_B14": "14E",
00101                 "RSL_CONFIG_DATA_B15": "14F",
00102                 "RSL_CONFIG_DATA_B16": "150",
00103                 "RSL_CONFIG_DATA_B17": "151",
00104                 "RSL_CONFIG_DATA_B18": "152",
00105                 "RSL_CONFIG_DATA_B19": "153",
00106                 "RSL_CONFIG_DATA_B20": "154",
00107                 "RSL_CONFIG_DATA_B21": "155",
00108                 "RSL_CONFIG_DATA_B22": "156",
00109                 "RSL_CONFIG_DATA_B23": "157",
00110                 "RSL_CONFIG_DATA_B24": "158",
00111                 "RSL_CONFIG_DATA_B25": "159",
00112                 "RSL_CONFIG_DATA_B26": "15A",
00113                 "RSL_CONFIG_DATA_B27": "15B",
00114                 "RSL_CONFIG_DATA_B28": "15C",
00115                 "RSL_CONFIG_DATA_B29": "15D",
00116                 "RSL_CONFIG_DATA_B30": "15E",
00117                 "RSL_CONFIG_DATA_B31": "15F",
00118                 "RSL_CONFIG_DATA_A0": "160",
```

```

00119         "RSL_CONFIG_DATA_A1": "161",
00120         "RSL_CONFIG_DATA_A2": "162",
00121         "RSL_CONFIG_DATA_A3": "163",
00122         "RSL_CONFIG_DATA_A4": "164",
00123         "RSL_CONFIG_DATA_A5": "165",
00124         "RSL_CONFIG_DATA_A6": "166",
00125         "RSL_CONFIG_DATA_A7": "167",
00126         "RSL_CONFIG_DATA_A8": "168",
00127         "RSL_CONFIG_DATA_A9": "169",
00128         "RSL_CONFIG_DATA_A10": "16A",
00129         "RSL_CONFIG_DATA_A11": "16B",
00130         "RSL_CONFIG_DATA_A12": "16C",
00131         "RSL_CONFIG_DATA_A13": "16D",
00132         "RSL_CONFIG_DATA_A14": "16E",
00133         "RSL_CONFIG_DATA_A15": "16F",
00134         "RSL_CONFIG_DATA_A16": "170",
00135         "RSL_CONFIG_DATA_A17": "171",
00136         "RSL_CONFIG_DATA_A18": "172",
00137         "RSL_CONFIG_DATA_A19": "173",
00138         "RSL_CONFIG_DATA_A20": "174",
00139         "RSL_CONFIG_DATA_A21": "175",
00140         "RSL_CONFIG_DATA_A22": "176",
00141         "RSL_CONFIG_DATA_A23": "177",
00142         "RSL_CONFIG_DATA_A24": "178",
00143         "RSL_CONFIG_DATA_A25": "179",
00144         "RSL_CONFIG_DATA_A26": "17A",
00145         "RSL_CONFIG_DATA_A27": "17B",
00146         "RSL_CONFIG_DATA_A28": "17C",
00147         "RSL_CONFIG_DATA_A29": "17D",
00148         "RSL_CONFIG_DATA_A30": "17E",
00149         "RSL_CONFIG_DATA_A31": "17F",
00150     }
00151 )
00152
00153     self.sens_subregisters = [
00154
00155         ("HST_MODE", "HS_TIMING_CTL", 0, 1, True),
00156         ("SLOWREADOFF_0", "CTRL_REG", 4, 1, True),
00157         ("SLOWREADOFF_1", "CTRL_REG", 5, 1, True),
00158         ("MANSHUT_MODE", "CTRL_REG", 8, 1, True),
00159         ("INTERLACING_EN", "CTRL_REG", 9, 1, True),
00160         ("HFW", "RSL_HFW_MODE_EN", 0, 1, True),
00161         ("ZDT_A", "RSL_ZDT_MODE_A_EN", 0, 1, True),
00162         ("ZDT_B", "RSL_ZDT_MODE_B_EN", 0, 1, True),
00163         ("HST_DEL_EN", "HST_DELAY_EN", 0, 1, True),
00164         ("PHI_DELAY_A", "HST_PHI_DELAY_DATA", 9, 10, True),
00165         ("PHI_DELAY_B", "HST_PHI_DELAY_DATA", 29, 10, True),
00166         # Assume that daedalus is not to be used with v1 board
00167         ("VRESET_HIGH", "VRESET_HIGH_VALUE", 15, 16, True),
00168
00169         ("STAT_SHORISEUR", "STAT_REG", 3, 1, False),
00170         ("STAT_SHOFALLUR", "STAT_REG", 4, 1, False),
00171         ("STAT_RSLNALLWENA", "STAT_REG", 12, 1, False),
00172         ("STAT_RSLNALLWENB", "STAT_REG", 15, 1, False),
00173         # ("STAT_CONFIGHSTDONE", "STAT_REG", 16, 1, False),
00174     ]
00175
00176
00177
00178

```

6.2.3 Member Function Documentation

6.2.3.1 getManualTiming()

```

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

```

Read off manual shutter timing settings

Returns:

list of manual timing intervals

Reimplemented from [nsCamera.sensors.sensorBase.sensorBase](#).

Definition at line 659 of file [daedalus.py](#).

```
00659     def getManualTiming(self):
00660         """
00661         Read off manual shutter timing settings
00662         Returns:
00663             list of manual timing intervals
00664         """
00665         timing = []
00666         for reg in [
00667             "EXT_PHI_CLK_SH0_ON",
00668             "EXT_PHI_CLK_SH0_OFF",
00669             "EXT_PHI_CLK_SH1_ON",
00670             "EXT_PHI_CLK_SH1_OFF",
00671             "EXT_PHI_CLK_SH2_ON",
00672         ]:
00673             _, reghex = self.ca.getRegister(reg)
00674             timing.append(25 * int(reghex, 16))
00675         return timing
00676
```

6.2.3.2 getSensTemp()

```
nsCamera.sensors.daedalus.daedalus.getSensTemp (
    self,
    scale = None,
    offset = None,
    slope = None,
    dec = 1 )
```

Read temperature sensor located on the Daedalus sensor

Args:

scale: temperature scale to report (defaults to C, options are F and K)
offset: offset of linear fit of sensor response (defaults to self.toffset)
slope: slope of linear fit of sensor response (defaults to self.tslope)
dec: round to 'dec' digits after the decimal point

Returns:

temperature as float on given scale, rounded to .1 degree

Reimplemented from [nsCamera.sensors.sensorBase.sensorBase](#).

Definition at line 677 of file [daedalus.py](#).

```
00677     def getSensTemp(self, scale=None, offset=None, slope=None, dec=1):
00678         """
00679         Read temperature sensor located on the Daedalus sensor
00680         Args:
00681             scale: temperature scale to report (defaults to C, options are F and K)
00682             offset: offset of linear fit of sensor response (defaults to self.toffset)
00683             slope: slope of linear fit of sensor response (defaults to self.tslope)
00684             dec: round to 'dec' digits after the decimal point
00685
00686         Returns:
00687             temperature as float on given scale, rounded to .1 degree
00688         """
00689         err, rval = self.ca.getMonV("MON_TSENSE_OUT", errflag=True)
00690         if err:
00691             logging.error(
00692                 self.logerr + "unable to retrieve temperature information ("
00693                     'getTemp'), returning "0" '
00694             )
00695             return 0.0
00696         if offset is None:
00697             offset = self.toffset
00698         if slope is None:
00699             slope = self.tslope
```

```

00700
00701         ctemp = offset + slope * rval
00702         if scale == "K":
00703             temp = round(ctemp + 273.15, dec)
00704         elif scale == "F":
00705             temp = round(1.8 * ctemp + 32, dec)
00706         else:
00707             temp = round(ctemp, dec)
00708         return temp
00709

```

6.2.3.3 parseReadoff()

```

nsCamera.sensors.daedalus.daedalus.parseReadoff (
    self,
    frames,
    columns )

```

Parses frames from board into images

Args:

frames: list of data arrays (frames) returned from board
columns: 1 (full width image) or 2 (hemispheres generate distinct images)

Returns:

list of data arrays (frames) reordered and deinterlaced

Reimplemented from [nsCamera.sensors.sensorBase.sensorBase](#).

Definition at line 710 of file [daedalus.py](#).

```

00710     def parseReadoff(self, frames, columns):
00711         """
00712         Parses frames from board into images
00713         Args:
00714             frames: list of data arrays (frames) returned from board
00715             columns: 1 (full width image) or 2 (hemispheres generate distinct images)
00716         Returns:
00717             list of data arrays (frames) reordered and deinterlaced
00718         """
00719         logging.debug(self.logdebug + "parseReadoff")
00720         w = self.width
00721         if hasattr(self, "ca"): # TODO: this may no longer be necessary
00722             padIt = self.ca.padToFull
00723         else:
00724             padIt = self.padToFull
00725         if padIt:
00726             rows = self.maxheight
00727         else:
00728             rows = self.lastrow - self.firstrow + 1
00729         parsed = []
00730         for frame in frames:
00731             current = np.zeros((rows, w), dtype=np.uint16)
00732             mapped = np.zeros((rows, w), dtype=np.uint16)
00733             frame = frame.reshape(rows, w)
00734
00735             for entry in range(int(w / 2)):
00736                 col = 32 * (entry % 8) + entry // 8 # lookup from daedlookup.xls
00737                 for row in range(rows):
00738                     current[row][col] = frame[row][2 * entry]
00739                     current[row][col + 256] = frame[row][2 * entry + 1]
00740
00741             for row in range(rows):
00742                 mapped[row][0:32] = current[row][320:352]
00743                 mapped[row][32:64] = current[row][352:384]
00744                 mapped[row][64:96] = current[row][192:224]
00745                 mapped[row][96:128] = current[row][160:192]
00746                 mapped[row][128:160] = current[row][256:288]
00747                 mapped[row][160:192] = current[row][288:320]
00748                 mapped[row][192:224] = current[row][416:448]
00749                 mapped[row][224:256] = current[row][32:64]
00750                 mapped[row][256:288] = current[row][128:160]

```

```

00751         mapped[row][288:320] = current[row][224:256]
00752         mapped[row][320:352] = current[row][384:416]
00753         mapped[row][352:384] = current[row][448:480]
00754         mapped[row][384:416] = current[row][480:512]
00755         mapped[row][416:448] = current[row][0:32]
00756         mapped[row][448:480] = current[row][64:96]
00757         mapped[row][480:512] = current[row][96:128]
00758         parsed.append(mapped)
00759
00760     images = self.ca.partition(parsed, columns)
00761     flatimages = [flattenlist(x) for x in images]
00762     return flatimages
00763

```

6.2.3.4 reportStatusSensor()

```

nsCamera.sensors.daedalus.daedalus.reportStatusSensor (
    self,
    statusbits,
    statusbits2 )

```

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

Args:

- statusbits: result of checkStatus()
- statusbits2: result of checkStatus2()

Reimplemented from [nsCamera.sensors.sensorBase.sensorBase](#).

Definition at line 764 of file [daedalus.py](#).

```

00764     def reportStatusSensor(self, statusbits, statusbits2):
00765         """
00766         Print status messages from sensor-specific bits of status register or object
00767         status flags
00768
00769         Args:
00770             statusbits: result of checkStatus()
00771             statusbits2: result of checkStatus2()
00772         """
00773         if int(statusbits[3]):
00774             print(self.loginfo + "SH0_rise_B_edge detected")
00775         if int(statusbits[4]):
00776             print(self.loginfo + "SH0_fall_B_edge detected")
00777         if int(statusbits[12]):
00778             print(self.loginfo + "RSLNALLWENB detected")
00779         if int(statusbits[15]):
00780             print(self.loginfo + "RSLNALLWENA detected")
00781         if self.HFW:
00782             print(self.loginfo + "High Full Well mode active")
00783         # TODO: handle two hemispheres for ZDT
00784         elif self.ZDT:
00785             print(self.loginfo + "Zero Dead Time mode active")
00786         elif self.interlacing != [0, 0]:
00787             print(
00788                 "{loginfo}Interlacing active: {interlacing}".format(
00789                     loginfo=self.loginfo, interlacing=str(self.interlacing)
00790                 )
00791             )
00792         if self.ca.sensmanual == []:
00793             print(
00794                 "{loginfo}High-speed timing: A:{Atiming}, B:{Btiming}".format(
00795                     loginfo=self.loginfo,
00796                     Atiming=self.getTiming(side="A", actual=True),
00797                     Btiming=self.getTiming(side="B", actual=True),
00798                 )
00799             )
00800         else:
00801             print(

```

```

00802         "{loginfo}Manual timing set to {timing}".format(
00803             loginfo=self.logininfo, timing=self.getManualTiming()
00804         )
00805     )
00806
00807
00808 """
00809 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00810 LLNL-CODE-838080
00811
00812 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00813 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00814 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00815 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00816 be made under this license.
00817 """

```

6.2.3.5 selectOscillator()

```

nsCamera.sensors.daedalus.daedalus.selectOscillator (
    self,
    osc = None )

```

Selects oscillator to control sensor timing

Args:

osc: 500|100|'ring'|external', defaults to 500 MHz

Returns:

error message as string

Reimplemented from [nsCamera.sensors.sensorBase.sensorBase](#).

Definition at line 444 of file [daedalus.py](#).

```

00444     def selectOscillator(self, osc=None):
00445         """
00446         Selects oscillator to control sensor timing
00447         Args:
00448             osc: 500|100|'ring'|external', defaults to 500 MHz
00449
00450         Returns:
00451             error message as string
00452         """
00453         logging.info(self.logininfo + "selectOscillator; osc = " + str(osc))
00454         if osc is None:
00455             osc = 500
00456         osc = str(osc)
00457         if osc[:3] == "500":
00458             payload = "00"
00459         elif osc[:3] == "100":
00460             payload = "01"
00461         elif osc.upper()[3] == "RIN":
00462             payload = "10"
00463         elif osc.upper()[3] in ["EXT"]:
00464             payload = "11"
00465         else:
00466             err = (
00467                 self.logerr + "selectOscillator: invalid parameter supplied. "
00468                 "Oscillator selection is unchanged."
00469             )
00470             logging.error(err)
00471             return err
00472         self.ca.setSubregister("OSC_SELECT", payload)
00473

```


6.2.3.6 sensorSpecific()

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

Returns:

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

Definition at line 180 of file [daedalus.py](#).

```
00180     def sensorSpecific(self):
00181         """
00182         Returns:
00183             list of tuples, (Sensor-specific register, default setting)
00184         """
00185         return [
00186             ("FPA_FRAME_INITIAL", "00000000"),
00187             ("FPA_FRAME_FINAL", "00000002"),
00188             ("FPA_ROW_INITIAL", "00000000"),
00189             ("FPA_ROW_FINAL", "000003FF"),
00190             ("HS_TIMING_DATA_ALO", "00006666"), # 0db6 = 2-1; 6666 = 2-2
00191             ("HS_TIMING_DATA_AHI", "00000000"),
00192             ("HS_TIMING_DATA_BLO", "00006666"),
00193             ("HS_TIMING_DATA_BHI", "00000000"),
00194             ("FRAME_ORDER_SEL", "00000000"),
00195             ("RSL_HFW_MODE_EN", "00000000"),
00196             ("RSL_ZDT_MODE_B_EN", "00000000"),
00197             ("RSL_ZDT_MODE_A_EN", "00000000"),
00198             ("RSL_CONFIG_DATA_B0", "00000000"),
00199             ("RSL_CONFIG_DATA_B1", "00000000"),
00200             ("RSL_CONFIG_DATA_B2", "00000000"),
00201             ("RSL_CONFIG_DATA_B3", "00000000"),
00202             ("RSL_CONFIG_DATA_B4", "00000000"),
00203             ("RSL_CONFIG_DATA_B5", "00000000"),
00204             ("RSL_CONFIG_DATA_B6", "00000000"),
00205             ("RSL_CONFIG_DATA_B7", "00000000"),
00206             ("RSL_CONFIG_DATA_B8", "00000000"),
00207             ("RSL_CONFIG_DATA_B9", "00000000"),
00208             ("RSL_CONFIG_DATA_B10", "00000000"),
00209             ("RSL_CONFIG_DATA_B11", "00000000"),
00210             ("RSL_CONFIG_DATA_B12", "00000000"),
00211             ("RSL_CONFIG_DATA_B13", "00000000"),
00212             ("RSL_CONFIG_DATA_B14", "00000000"),
00213             ("RSL_CONFIG_DATA_B15", "00000000"),
00214             ("RSL_CONFIG_DATA_B16", "00000000"),
00215             ("RSL_CONFIG_DATA_B17", "00000000"),
00216             ("RSL_CONFIG_DATA_B18", "00000000"),
00217             ("RSL_CONFIG_DATA_B19", "00000000"),
00218             ("RSL_CONFIG_DATA_B20", "00000000"),
00219             ("RSL_CONFIG_DATA_B21", "00000000"),
00220             ("RSL_CONFIG_DATA_B22", "00000000"),
00221             ("RSL_CONFIG_DATA_B23", "00000000"),
00222             ("RSL_CONFIG_DATA_B24", "00000000"),
00223             ("RSL_CONFIG_DATA_B25", "00000000"),
00224             ("RSL_CONFIG_DATA_B26", "00000000"),
00225             ("RSL_CONFIG_DATA_B27", "00000000"),
00226             ("RSL_CONFIG_DATA_B28", "00000000"),
00227             ("RSL_CONFIG_DATA_B29", "00000000"),
00228             ("RSL_CONFIG_DATA_B30", "00000000"),
00229             ("RSL_CONFIG_DATA_B31", "00000000"),
00230             ("RSL_CONFIG_DATA_A0", "00000000"),
00231             ("RSL_CONFIG_DATA_A1", "00000000"),
00232             ("RSL_CONFIG_DATA_A2", "00000000"),
00233             ("RSL_CONFIG_DATA_A3", "00000000"),
00234             ("RSL_CONFIG_DATA_A4", "00000000"),
00235             ("RSL_CONFIG_DATA_A5", "00000000"),
00236             ("RSL_CONFIG_DATA_A6", "00000000"),
00237             ("RSL_CONFIG_DATA_A7", "00000000"),
00238             ("RSL_CONFIG_DATA_A8", "00000000"),
00239             ("RSL_CONFIG_DATA_A9", "00000000"),
00240             ("RSL_CONFIG_DATA_A10", "00000000"),
00241             ("RSL_CONFIG_DATA_A11", "00000000"),
00242             ("RSL_CONFIG_DATA_A12", "00000000"),
00243             ("RSL_CONFIG_DATA_A13", "00000000"),
00244             ("RSL_CONFIG_DATA_A14", "00000000"),
00245             ("RSL_CONFIG_DATA_A15", "00000000"),
```

```

00246         ("RSL_CONFIG_DATA_A16", "00000000"),
00247         ("RSL_CONFIG_DATA_A17", "00000000"),
00248         ("RSL_CONFIG_DATA_A18", "00000000"),
00249         ("RSL_CONFIG_DATA_A19", "00000000"),
00250         ("RSL_CONFIG_DATA_A20", "00000000"),
00251         ("RSL_CONFIG_DATA_A21", "00000000"),
00252         ("RSL_CONFIG_DATA_A22", "00000000"),
00253         ("RSL_CONFIG_DATA_A23", "00000000"),
00254         ("RSL_CONFIG_DATA_A24", "00000000"),
00255         ("RSL_CONFIG_DATA_A25", "00000000"),
00256         ("RSL_CONFIG_DATA_A26", "00000000"),
00257         ("RSL_CONFIG_DATA_A27", "00000000"),
00258         ("RSL_CONFIG_DATA_A28", "00000000"),
00259         ("RSL_CONFIG_DATA_A29", "00000000"),
00260         ("RSL_CONFIG_DATA_A30", "00000000"),
00261         ("RSL_CONFIG_DATA_A31", "00000000"),
00262         ("HST_TRIGGER_DELAY_DATA_LO", "00000000"),
00263         ("HST_TRIGGER_DELAY_DATA_HI", "00000000"),
00264         ("HST_PHI_DELAY_DATA", "00000000"),
00265         ("SLOWREADOFF_0", "0"),
00266         ("SLOWREADOFF_1", "0"),
00267     ]
00268

```

6.2.3.7 setExtClk()

```

nsCamera.sensors.daedalus.daedalus.setExtClk (
    self,
    dilation = None,
    frequency = None )

```

Override the standard board clock with the external clock.

Args:

dilation: ratio of base frequency (500 MHz) to desired external clock
 frequency: Default is 25. Overridden if frequency parameter is provided
 frequency: Desired frequency for phi clock.

Returns:

error message as string

Reimplemented from [nsCamera.sensors.sensorBase.sensorBase](#).

Definition at line 556 of file [daedalus.py](#).

```

00556     def setExtClk(self, dilation=None, frequency=None):
00557         """
00558         Override the standard board clock with the external clock.
00559         Args:
00560             dilation: ratio of base frequency (500 MHz) to desired external clock
00561             frequency: Default is 25. Overridden if frequency parameter is provided
00562             frequency: Desired frequency for phi clock.
00563         Returns:
00564             error message as string
00565         """
00566         logging.debug(
00567             self.logdebug
00568             + "setExtClk; dilation = "
00569             + str(dilation)
00570             + "; frequency = "
00571             + str(frequency)
00572         )
00573         if not (isinstance(frequency, int) or isinstance(frequency, float)):
00574             err = (
00575                 self.logerr
00576                 + "invalid external clock frequency submitted. Clock is not "
00577                 + "operating"
00578             )
00579             logging.error(err)
00580             return err
00581         self.ca.selectOscillator("external")
00582         if not dilation:

```

```

00583         dilation = 25
00584     if not frequency:
00585         frequency = 5e7 / float(dilation)
00586     count = 2e7 / float(frequency) - 1 # base phi clock is 20 MHz?
00587     if count < 0:
00588         count = 0
00589         warn = (
00590             self.logwarn
00591             + "external clock frequency exceeding maximum. Frequency set to "
00592             + "maximum (20 MHz)"
00593         )
00594         logging.warning(warn)
00595     if count > 0xFFFFFFFF:
00596         count = 0xFFFFFFFF
00597     counthex = hex(int(count))[2:].zfill(8)
00598     self.ca.setRegister("HST_EXT_CLK_HALF_PER", counthex)
00599

```

6.2.3.8 setHighFullWell()

```

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 and column number will be reset to 0. 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

Reimplemented from [nsCamera.sensors.sensorBase.sensorBase](#).

Definition at line 354 of file [daedalus.py](#).

```

00354     def setHighFullWell(self, flag):
00355         """
00356         Activates High Full Well mode. All frames are acquired simultaneously. Zero Dead
00357         Time mode and interlacing will be automatically deactivated and column number
00358         will be reset to 0. NOTE: after deactivating HFW, the board remains in
00359         uninterlaced mode (interlacing = 0)
00360
00361         Args:
00362             flag: True to activate HFW mode, False to deactivate
00363
00364         Returns:
00365             Error message
00366         """
00367         logging.debug(self.logdebug + "setHighFullWell; flag = " + str(flag))
00368         err0 = ""
00369         if flag:
00370             if self.ZDT:
00371                 logging.warning(
00372                     self.logwarn + "ZDT mode will be disengaged because of HFW "
00373                     "setting "
00374                 )
00375                 err0 = self.setZeroDeadTime(False)
00376                 err1, _ = self.ca.setSubregister("HFW", "1")
00377                 self.HFW = False # preclude HFW deactivation message in setInterlacing
00378                 self.setInterlacing(0)
00379                 self.HFW = True
00380                 logging.info(self.loginfo + "High Full Well mode active")
00381             else:
00382                 self.HFW = False
00383                 err1, _ = self.ca.setSubregister("HFW", "0")
00384                 logging.info(self.loginfo + "High Full Well mode inactivate")
00385             err = err0 + err1
00386             if err:
00387                 logging.error(self.logerr + "HFW option may not be set correctly ")
00388         return err
00389

```

6.2.3.9 setInterlacing()

```
nsCamera.sensors.daedalus.daedalus.setInterlacing (
    self,
    ifactor = None,
    side = None )
```

Sets interlacing factor. NOTE: if called directly when HFW or ZDT mode is active, this will disengage those modes automatically. If hemispheres have different factors when the image is acquired, the resulting frames are separated into half-width images

Args:

ifactor: number of interlaced lines (generates ifactor + 1 images per frame)
 defaults to 0 (no interlacing)
 side: identify particular hemisphere (A or B) to control. If left blank,
 control both hemispheres

Returns:

integer: active interlacing factor (unchanged if error)

Reimplemented from [nsCamera.sensors.sensorBase.sensorBase](#).

Definition at line 269 of file [daedalus.py](#).

```
00269     def setInterlacing(self, ifactor=None, side=None):
00270         """
00271         Sets interlacing factor. NOTE: if called directly when HFW or ZDT mode is active,
00272         this will disengage those modes automatically. If hemispheres have different
00273         factors when the image is acquired, the resulting frames are separated into
00274         half-width images
00275
00276         Args:
00277             ifactor: number of interlaced lines (generates ifactor + 1 images per frame)
00278             defaults to 0 (no interlacing)
00279             side: identify particular hemisphere (A or B) to control. If left blank,
00280             control both hemispheres
00281
00282         Returns:
00283             integer: active interlacing factor (unchanged if error)
00284         """
00285         logging.debug(self.logdebug + "setInterlacing; ifactor = " + str(ifactor))
00286         if ifactor is None:
00287             ifactor = 0
00288         if (
00289             not isinstance(ifactor, int)
00290             or ifactor < 0
00291             or ifactor > (self.maxheight - 1)
00292         ):
00293             err = (
00294                 self.logerr + "invalid interlacing factor submitted. "
00295                 "Interlacing remains unchanged. "
00296             )
00297             logging.error(err)
00298             return self.interlacing
00299         if self.HFW:
00300             logging.warning(
00301                 self.logwarn + "HFW mode will be disengaged because of new "
00302                 "interlacing setting "
00303             )
00304             self.setHighFullWell(False)
00305         if self.ZDT:
00306             logging.warning(
00307                 self.logwarn + "ZDT mode will be disengaged because of new "
00308                 "interlacing setting "
00309             )
00310             self.setZeroDeadTime(False)
00311         if ifactor == 0:
00312             bitscheme = self.maxheight * [0]
00313             # deactivating one side shouldn't turn off enable for both sides
00314             # TODO: is it a problem if sides are set separately, so interlacing is zero
00315             # but still enabled?
```

```

00316         if side is None:
00317             self.ca.setSubregister("INTERLACING_EN", "0")
00318         else:
00319             pattern = [0] + ifactor * [1]
00320             reps = 1 + self.maxheight // (ifactor + 1)
00321             bitscheme = (reps * pattern)[0 : self.maxheight]
00322             self.ca.setSubregister("INTERLACING_EN", "1")
00323         err = ""
00324         for regnum in range(32):
00325             regbits = bitscheme[32 * regnum : 32 * (regnum + 1)]
00326             logging.debug(self.logdebug + "regbits = " + str(regbits))
00327             # generated pattern is reverse order from placement in register (element 0
00328             # of the list is the LSB of the register)
00329             bitsrev = regbits[::-1]
00330             s = [str(i) for i in bitsrev]
00331             b = "".join(s) # assemble as binary number for processing
00332             hexval = "%x" % int(b, 2)
00333             val = hexval.zfill(8)
00334             err0 = ""
00335             err1 = ""
00336             if side is None or side.lower() == "a":
00337                 lname = "RSL_CONFIG_DATA_A" + str(regnum)
00338                 err1, _ = self.ca.setRegister(lname, val)
00339                 self.interlacing[1] = ifactor
00340             if side is None or side.lower() == "b":
00341                 rname = "RSL_CONFIG_DATA_B" + str(regnum)
00342                 err0, _ = self.ca.setRegister(rname, val)
00343                 self.interlacing[0] = ifactor
00344             err = err + err0 + err1
00345         if err:
00346             logging.error(self.logerr + "interlacing may not be set correctly: " + err)
00347         logging.info(self.loginfo + "Interlacing set to " + str(self.interlacing))
00348         if self.interlacing[0] == self.interlacing[1]:
00349             self.columns = 1
00350         else:
00351             self.columns = 2
00352         return self.interlacing
00353

```

6.2.3.10 setManualShutters()

```

nsCamera.sensors.daedalus.daedalus.setManualShutters (
    self,
    timing = None )

```

Legacy alias for setManualTiming()

Reimplemented from [nsCamera.sensors.sensorBase.sensorBase](#).

Definition at line 602 of file [daedalus.py](#).

```

00602     def setManualShutters(self, timing=None):
00603         """
00604         Legacy alias for setManualTiming()
00605         """
00606         self.setManualTiming(timing)
00607

```

6.2.3.11 setManualTiming()

```

nsCamera.sensors.daedalus.daedalus.setManualTiming (
    self,
    timing = None )

```

Manual shutter timing, five intervals given in nanoseconds, e.g.,
 [100,50,100,50,100] for frame 0 open for 100 ns, an interframe pause of 50 ns,
 frame 1 open for 100 ns, etc. Timing is set for both hemispheres.

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

- Minimum timing is 75 ns
- Maximum is $25 * 2^{30}$ ns (approximately 27 seconds)

Args:

timing: 5-element list in nanoseconds

Returns:

tuple (error string, response string from final message)

Reimplemented from [nsCamera.sensors.sensorBase.sensorBase](#).

Definition at line 608 of file [daedalus.py](#).

```
00608     def setManualTiming(self, timing=None):
00609         """
00610         Manual shutter timing, five intervals given in nanoseconds, e.g.,
00611         [100,50,100,50,100] for frame 0 open for 100 ns, an interframe pause of 50 ns,
00612         frame 1 open for 100 ns, etc. Timing is set for both hemispheres.
00613
00614         The actual timing is rounded down to the nearest multiple of 25 ns. (Each
00615         count = 25 ns. e.g., a request for 140 ns rounds down to a count of '5',
00616         which corresponds to 125 ns))
00617         - Minimum timing is 75 ns
00618         - Maximum is 25 * 2^30 ns (approximately 27 seconds)
00619
00620         Args:
00621         timing: 5-element list in nanoseconds
00622
00623         Returns:
00624         tuple (error string, response string from final message)
00625         """
00626         if timing is None:
00627             logging.info(
00628                 self.loginfo
00629                 + "No manual timing setting provided, defaulting to (100, 150, 100, "
00630                 + "150, 100, 150, 100) for both hemispheres"
00631             )
00632             timing = [(100, 150, 100, 150, 100)]
00633
00634             logging.info(self.loginfo + "Manual shutter sequence: " + str(timing))
00635             flattened = flattenList(timing)
00636             if (
00637                 len(flattened) != 5
00638                 or not all(isinstance(x, (int, float)) for x in flattened)
00639                 or not all(x >= 25 for x in flattened)
00640             ):
00641                 err = self.logerr + "Invalid manual shutter timing list: " + str(timing)
00642                 logging.error(err + "; timing settings unchanged")
00643                 return err, "00000000"
00644
00645             timecounts = [int(a // 25) for a in flattened]
00646             self.ca.sensmanual = timing
00647             self.ca.senstiming = {} # clear HST settings from ca object
00648
00649             control_messages = [
00650                 ("MANSHUT_MODE", "1"),
00651                 ("EXT_PHI_CLK_SH0_ON", "{0:#0{1}x}".format(timecounts[0], 10)[2:10]),
00652                 ("EXT_PHI_CLK_SH0_OFF", "{0:#0{1}x}".format(timecounts[1], 10)[2:10]),
00653                 ("EXT_PHI_CLK_SH1_ON", "{0:#0{1}x}".format(timecounts[2], 10)[2:10]),
00654                 ("EXT_PHI_CLK_SH1_OFF", "{0:#0{1}x}".format(timecounts[3], 10)[2:10]),
00655                 ("EXT_PHI_CLK_SH2_ON", "{0:#0{1}x}".format(timecounts[4], 10)[2:10]),
00656             ]
00657             return self.ca.submitMessages(control_messages, " setManualShutters: ")
00658
```

6.2.3.12 setPhiDelay()

nsCamera.sensors.daedalus.daedalus.setPhiDelay (

```

        self,
        side = None,
        delay = 0 )

```

Use phi delay timer. Actual delay is rounded down to multiple of .15 ns, up to a maximum delay of 1.5 ns

Args:

side: hemisphere to delay; if None, delay both hemispheres
 delay: phi delay in ns

Returns:

String of errors, if any

Reimplemented from [nsCamera.sensors.sensorBase.sensorBase](#).

Definition at line 516 of file [daedalus.py](#).

```

00516     def setPhiDelay(self, side=None, delay=0):
00517         """
00518         Use phi delay timer. Actual delay is rounded down to multiple of .15 ns, up to a
00519         maximum delay of 1.5 ns
00520         Args:
00521             side: hemisphere to delay; if None, delay both hemispheres
00522             delay: phi delay in ns
00523
00524         Returns:
00525             String of errors, if any
00526         """
00527         logging.debug(self.logdebug + "setPhiDelay; delay = " + str(delay))
00528         if (
00529             not (isinstance(delay, int) or isinstance(delay, float))
00530             or delay < 0
00531             or delay > 1.5
00532         ):
00533             err = (
00534                 self.logerr + "invalid phi delay submitted. Delay remains "
00535                 "unchanged. "
00536             )
00537             logging.error(err)
00538             return err
00539         delayblocks = int(delay / 0.15)
00540         if delayblocks < 0:
00541             delayblocks = 0
00542         if delayblocks > 10:
00543             delayblocks = 10
00544         delayseq = (10 - delayblocks) * [0] + delayblocks * [1]
00545         seqstr = "".join(str(x) for x in delayseq)
00546         err1 = ""
00547         err2 = ""
00548         if side is None or side.upper() == "A":
00549             err1, _ = self.ca.setSubregister("PHI_DELAY_A", seqstr)
00550         if side is None or side.upper() == "B":
00551             err2, _ = self.ca.setSubregister("PHI_DELAY_B", seqstr)
00552         delayed = delayblocks * 0.15
00553         logging.info(self.loginfo + "Actual phi delay = " + str(delayed) + " ns")
00554         return err1 + err2
00555

```

6.2.3.13 setTriggerDelay()

```

nsCamera.sensors.daedalus.daedalus.setTriggerDelay (
    self,
    delay = 0 )

```

Use trigger delay timer. Actual delay is rounded down to multiple of .15 ns, up to a maximum delay of 6 ns

Args:
 delay: trigger delay in ns

Returns:
 String of errors, if any

Reimplemented from [nsCamera.sensors.sensorBase.sensorBase](#).

Definition at line 474 of file [daedalus.py](#).

```

00474     def setTriggerDelay(self, delay=0):
00475         """
00476         Use trigger delay timer. Actual delay is rounded down to multiple of .15 ns, up
00477         to a maximum delay of 6 ns
00478
00479         Args:
00480             delay: trigger delay in ns
00481
00482         Returns:
00483             String of errors, if any
00484         """
00485         logging.debug(self.logdebug + "setTriggerDelay; delay = " + str(delay))
00486         if (
00487             not (isinstance(delay, int) or isinstance(delay, float))
00488             or delay < 0
00489             or delay > 6
00490         ):
00491             err = (
00492                 self.logerr + "invalid trigger delay submitted. Delay remains "
00493                 "unchanged. "
00494             )
00495             logging.error(err)
00496             return err
00497         delayblocks = int(delay / 0.15)
00498         if delayblocks < 0:
00499             delayblocks = 0
00500         if delayblocks > 40:
00501             delayblocks = 40
00502         delayseq = (40 - delayblocks) * [0] + delayblocks * [1]
00503         seqstr = "".join(str(x) for x in delayseq)
00504         seqhex = "%x" % int(seqstr, 2)
00505         logging.debug(self.logdebug + "seqhex = " + str(seqhex))
00506         highpart = seqhex[-10:-8].zfill(8)
00507         lowpart = seqhex[-8:].zfill(8)
00508         self.ca.setSubregister("HST_DEL_EN", "1")
00509         err0, _ = self.ca.setRegister("HST_TRIGGER_DELAY_DATA_LO", lowpart)
00510         err1, _ = self.ca.setRegister("HST_TRIGGER_DELAY_DATA_HI", highpart)
00511         err2, _ = self.ca.setSubregister("HST_MODE", "1")
00512         delayed = delayblocks * 0.15
00513         logging.info(self.loginfo + "Actual trigger delay = " + str(delayed) + " ns")
00514         return err0 + err1 + err2
00515

```

6.2.3.14 setZeroDeadTime()

```

nsCamera.sensors.daedalus.daedalus.setZeroDeadTime (
    self,
    flag = True,
    side = None )

```

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
 side: identify particular hemisphere (A or B) to control. If left blank,

control both hemispheres

Returns:

Error message

Reimplemented from [nsCamera.sensors.sensorBase.sensorBase](#).

Definition at line 390 of file [daedalus.py](#).

```

00390     def setZeroDeadTime(self, flag=True, side=None):
00391         """
00392         Activates Zero Dead Time mode. Even rows follow the assigned HST schedule; odd
00393         rows are acquired while the 'shutter' for the even rows are closed. High Full
00394         Well mode and interlacing will be automatically deactivated.
00395         *NOTE* after deactivating ZDT, the board reverts to uninterlaced mode
00396         (interlacing = 0)
00397
00398         Args:
00399             flag: True to activate ZDT mode, False to deactivate
00400             side: identify particular hemisphere (A or B) to control. If left blank,
00401                 control both hemispheres
00402
00403         Returns:
00404             Error message
00405         """
00406         logging.debug(self.logdebug + "setZeroDeadTime; flag = " + str(flag))
00407         err0 = ""
00408         err1 = ""
00409         err2 = ""
00410         if flag:
00411             if self.HFW:
00412                 logging.warning(
00413                     self.logwarn + "HFW mode will be disengaged because of ZDT "
00414                     "setting "
00415                 )
00416                 err0 = self.setHighFullWell(False)
00417             if side is None or side.lower() == "a":
00418                 err2, _ = self.ca.setSubregister("ZDT_A", "1")
00419                 self.interlacing[0] = 1
00420             if side is None or side.lower() == "b":
00421                 err1, _ = self.ca.setSubregister("ZDT_B", "1")
00422                 self.interlacing[1] = 1
00423             # self.ZDT = False # preclude ZDT deactivation message in setInterlacing
00424             # if self.interlacing != [0, 0]:
00425             #     self.setInterlacing(0)
00426             # TODO: need to handle flags when ZDT active for just one side
00427             self.ZDT = True
00428             logging.info(
00429                 self.loginfo + "Zero Dead Time mode active; actual interlacing = 1"
00430             )
00431         else:
00432             self.ZDT = False
00433             if side is None or side.lower() == "a":
00434                 err2, _ = self.ca.setSubregister("ZDT_A", "0")
00435             if side is None or side.lower() == "b":
00436                 err1, _ = self.ca.setSubregister("ZDT_B", "0")
00437             self.setInterlacing(0)
00438             logging.info(self.loginfo + "Zero Dead Time mode inactivate")
00439         err = err0 + err1 + err2
00440         if err:
00441             logging.error(self.logerr + "ZDT option may not be set correctly ")
00442         return err
00443

```

6.2.4 Member Data Documentation

6.2.4.1 bytesperpixel

```
int nsCamera.sensors.daedalus.daedalus.bytesperpixel = 2 [static]
```

Definition at line 36 of file [daedalus.py](#).

6.2.4.2 ca

```
nsCamera.sensors.daedalus.daedalus.ca
```

Definition at line 57 of file [daedalus.py](#).

6.2.4.3 columns [1/2]

```
int nsCamera.sensors.daedalus.daedalus.columns = 1 [static]
```

Definition at line 51 of file [daedalus.py](#).

6.2.4.4 columns [2/2]

```
nsCamera.sensors.daedalus.daedalus.columns
```

Definition at line 349 of file [daedalus.py](#).

6.2.4.5 detect

```
str nsCamera.sensors.daedalus.daedalus.detect = "DAEDALUS_DET" [static]
```

Definition at line 38 of file [daedalus.py](#).

6.2.4.6 firstframe

```
int nsCamera.sensors.daedalus.daedalus.firstframe = 0 [static]
```

Definition at line 43 of file [daedalus.py](#).

6.2.4.7 firstrow

```
int nsCamera.sensors.daedalus.daedalus.firstrow = 0 [static]
```

Definition at line 48 of file [daedalus.py](#).

6.2.4.8 fpganumID

```
int nsCamera.sensors.daedalus.daedalus.fpganumID = 2 [static]
```

Definition at line 37 of file [daedalus.py](#).

6.2.4.9 height

```
int nsCamera.sensors.daedalus.daedalus.height = 1024 [static]
```

Definition at line 47 of file [daedalus.py](#).

6.2.4.10 HFW [1/2]

```
bool nsCamera.sensors.daedalus.daedalus.HFW = False [static]
```

Definition at line 42 of file [daedalus.py](#).

6.2.4.11 HFW [2/2]

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

Definition at line 377 of file [daedalus.py](#).

6.2.4.12 interlacing [1/2]

```
list nsCamera.sensors.daedalus.daedalus.interlacing = [0, 0] [static]
```

Definition at line 50 of file [daedalus.py](#).

6.2.4.13 interlacing [2/2]

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

Definition at line 347 of file [daedalus.py](#).

6.2.4.14 lastframe

```
int nsCamera.sensors.daedalus.daedalus.lastframe = 2 [static]
```

Definition at line 44 of file [daedalus.py](#).

6.2.4.15 lastrow

```
int nsCamera.sensors.daedalus.daedalus.lastrow = 1023 [static]
```

Definition at line 49 of file [daedalus.py](#).

6.2.4.16 loglabel

```
str nsCamera.sensors.daedalus.daedalus.loglabel = "[Daedalus] " [static]
```

Definition at line 40 of file [daedalus.py](#).

6.2.4.17 maxframe

```
int nsCamera.sensors.daedalus.daedalus.maxframe = 2 [static]
```

Definition at line 33 of file [daedalus.py](#).

6.2.4.18 maxheight

```
int nsCamera.sensors.daedalus.daedalus.maxheight = 1024 [static]
```

Definition at line 35 of file [daedalus.py](#).

6.2.4.19 maxwidth

```
int nsCamera.sensors.daedalus.daedalus.maxwidth = 512 [static]
```

Definition at line 34 of file [daedalus.py](#).

6.2.4.20 minframe

```
int nsCamera.sensors.daedalus.daedalus.minframe = 0 [static]
```

Definition at line 32 of file [daedalus.py](#).

6.2.4.21 nframes

```
int nsCamera.sensors.daedalus.daedalus.nframes = 3 [static]
```

Definition at line 45 of file [daedalus.py](#).

6.2.4.22 padToFull

```
bool nsCamera.sensors.daedalus.daedalus.padToFull = True [static]
```

Definition at line 52 of file [daedalus.py](#).

6.2.4.23 sens_registers

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

Definition at line 60 of file [daedalus.py](#).

6.2.4.24 sens_subregisters

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

Definition at line 153 of file [daedalus.py](#).

6.2.4.25 sensfam

```
str nsCamera.sensors.daedalus.daedalus.sensfam = "Daedalus" [static]
```

Definition at line 39 of file [daedalus.py](#).

6.2.4.26 specwarn

```
str nsCamera.sensors.daedalus.daedalus.specwarn = "" [static]
```

Definition at line 31 of file [daedalus.py](#).

6.2.4.27 toffset

```
float nsCamera.sensors.daedalus.daedalus.toffset = -165.76 [static]
```

Definition at line 53 of file [daedalus.py](#).

6.2.4.28 tslope

```
float nsCamera.sensors.daedalus.daedalus.tslope = 81.36 [static]
```

Definition at line 54 of file [daedalus.py](#).

6.2.4.29 width

```
int nsCamera.sensors.daedalus.daedalus.width = 512 [static]
```

Definition at line 46 of file [daedalus.py](#).

6.2.4.30 ZDT [1/2]

```
bool nsCamera.sensors.daedalus.daedalus.ZDT = False [static]
```

Definition at line 41 of file [daedalus.py](#).

6.2.4.31 ZDT [2/2]

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

Definition at line 427 of file [daedalus.py](#).

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

- C:/Users/hill35/git/camera_python/nsCamera/sensors/[daedalus.py](#)

6.3 nsCamera.utils.misc.fakeCA Class Reference

Public Member Functions

- [__init__](#) (self, [sensorname](#)="icarus2", firstframe=0, lastframe=3, width=512, height=1024, [padToFull](#)=True, firstrow=0, lastrow=1023, maxwidth=512, maxheight=1024, bytesperpixel=2, interlacing=None, columns=1, [logtag](#)=None)
- [partition](#) (self, parsed, columns)

Public Attributes

- [sensorname](#)
- [boardname](#)
- [padToFull](#)
- [logtag](#)
- [logcritbase](#)
- [logerrbase](#)
- [logwarnbase](#)
- [loginfobase](#)
- [logdebugbase](#)
- [logcrit](#)
- [logerr](#)
- [logwarn](#)
- [loginfo](#)
- [logdebug](#)
- [sensor](#)

6.3.1 Detailed Description

Fake 'cameraAssembler' object to use as a parameter object in offline functions.
Returned by generateFrames(), it contains the frame details required to save and plot images.

Definition at line 203 of file [misc.py](#).

6.3.2 Constructor & Destructor Documentation

6.3.2.1 __init__()

```
nsCamera.utils.misc.fakeCA.__init__ (
    self,
    sensorname = "icarus2",
    firstframe = 0,
    lastframe = 3,
    width = 512,
    height = 1024,
    padToFull = True,
    firstrow = 0,
    lastrow = 1023,
    maxwidth = 512,
    maxheight = 1024,
    bytesperpixel = 2,
    interlacing = None,
    columns = 1,
    logtag = None )
```

Definition at line 210 of file [misc.py](#).

```
00226     ):
00227         self.sensorname = sensorname
00228         self.boardname = None
00229         self.padToFull = padToFull
00230         if logtag is None:
00231             self.logtag = ""
00232         else:
00233             self.logtag = logtag
00234
00235         self.logcritbase = "CRITICAL" + self.logtag + ": "
00236         self.logerrbase = "ERROR" + self.logtag + ": "
00237         self.logwarnbase = "WARNING" + self.logtag + ": "
00238         self.loginfobase = "INFO" + self.logtag + ": "
00239         self.logdebugbase = "DEBUG" + self.logtag + ": "
00240
00241         self.logcrit = self.logcritbase + "[FS] "
00242         self.logerr = self.logerrbase + "[FS] "
00243         self.logwarn = self.logwarnbase + "[FS] "
00244         self.loginfo = self.loginfobase + "[FS] "
00245         self.logdebug = self.logdebugbase + "[FS] "
00246
00247         if self.sensorname == "icarus":
00248             import nsCamera.sensors.icarus as snsr
00249         elif self.sensorname == "icarus2":
00250             import nsCamera.sensors.icarus2 as snsr
00251         elif self.sensorname == "daedalus":
00252             import nsCamera.sensors.daedalus as snsr
00253
00254         self.sensor = snsr(self)
00255
```

6.3.3 Member Function Documentation

6.3.3.1 partition()

```
nsCamera.utils.misc.fakeCA.partition (  
    self,  
    parsed,  
    columns )
```

Definition at line 256 of file [misc.py](#).

```
00256     def partition(self, parsed, columns):  
00257         # return nsCamera.utils.misc.partition(self, parsed, columns)  
00258         return partition(self, parsed, columns)  
00259  
00260
```

6.3.4 Member Data Documentation

6.3.4.1 boardname

```
nsCamera.utils.misc.fakeCA.boardname
```

Definition at line 228 of file [misc.py](#).

6.3.4.2 logcrit

```
nsCamera.utils.misc.fakeCA.logcrit
```

Definition at line 241 of file [misc.py](#).

6.3.4.3 logcritbase

```
nsCamera.utils.misc.fakeCA.logcritbase
```

Definition at line 235 of file [misc.py](#).

6.3.4.4 logdebug

```
nsCamera.utils.misc.fakeCA.logdebug
```

Definition at line 245 of file [misc.py](#).

6.3.4.5 logdebugbase

```
nsCamera.utils.misc.fakeCA.logdebugbase
```

Definition at line 239 of file [misc.py](#).

6.3.4.6 logerr

`nsCamera.utils.misc.fakeCA.logerr`

Definition at line 242 of file [misc.py](#).

6.3.4.7 logerrbase

`nsCamera.utils.misc.fakeCA.logerrbase`

Definition at line 236 of file [misc.py](#).

6.3.4.8 loginfo

`nsCamera.utils.misc.fakeCA.loginfo`

Definition at line 244 of file [misc.py](#).

6.3.4.9 loginfobase

`nsCamera.utils.misc.fakeCA.loginfobase`

Definition at line 238 of file [misc.py](#).

6.3.4.10 logtag

`nsCamera.utils.misc.fakeCA.logtag`

Definition at line 231 of file [misc.py](#).

6.3.4.11 logwarn

`nsCamera.utils.misc.fakeCA.logwarn`

Definition at line 243 of file [misc.py](#).

6.3.4.12 logwarnbase

`nsCamera.utils.misc.fakeCA.logwarnbase`

Definition at line 237 of file [misc.py](#).

6.3.4.13 padToFull

`nsCamera.utils.misc.fakeCA.padToFull`

Definition at line 229 of file [misc.py](#).

6.3.4.14 sensor

`nsCamera.utils.misc.fakeCA.sensor`

Definition at line 254 of file [misc.py](#).

6.3.4.15 sensorname

`nsCamera.utils.misc.fakeCA.sensorname`

Definition at line 227 of file [misc.py](#).

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

- C:/Users/hill35/git/camera_python/nsCamera/utils/[misc.py](#)

6.4 nsCamera.comms.GigE.GigE Class Reference

Classes

- class [ZESTETM1_CARD_INFO](#)

Public Member Functions

- [__init__](#) (self, camassem)
- [sendCMD](#) (self, pkt)
- [arm](#) (self, [mode](#))
- [readFrames](#) (self, waitOnSRAM, timeout=0, fast=False, columns=1)
- [readoff](#) (self, waitOnSRAM, timeout=0, fast=False, columns=1)
- [writeSerial](#) (self, outstring, timeout)
- [readSerial](#) (self, size, timeout=None)
- [openDevice](#) (self, timeout=30)
- [closeDevice](#) (self)
- [getCardIP](#) (self)
- [getCardInfo](#) (self)

Public Attributes

- [ca](#)
- [logcrit](#)
- [logerr](#)
- [logwarn](#)
- [loginfo](#)
- [logdebug](#)
- [mode](#)
- [writeTimeout](#)
- [readTimeout](#)
- [payloadsize](#)
- [skipError](#)
- [ZErrorDict](#)
- [dport](#)
- [closecard](#)
- [CardInfo](#)
- [CardInfoP](#)
- [ZCountCards](#)
- [ZESTETM1_CARD_INFO](#)
- [ZOpenConnection](#)
- [ZWriteData](#)
- [ZReadData](#)
- [Connection](#)

Protected Attributes

- [_zest](#)

6.4.1 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
readFrames() - waits for data ready register flag, then copies camera image data
               into numpy arrays
readoff() - waits for data ready register flag, then copies camera image data
            into numpy arrays; returns payload, payload size, and error message
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 27 of file [GigE.py](#).

6.4.2 Constructor & Destructor Documentation

6.4.2.1 `__init__()`

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

Args:

camassem: parent cameraAssembler object

Definition at line 49 of file [GigE.py](#).

```
00049     def __init__(self, camassem):
00050         """
00051         Args:
00052             camassem: parent cameraAssembler object
00053         """
00054         self.ca = camassem
00055         self.logcrit = self.ca.logcritbase + "[GigE] "
00056         self.logerr = self.ca.logerrbase + "[GigE] "
00057         self.logwarn = self.ca.logwarnbase + "[GigE] "
00058         self.loginfo = self.ca.loginfobase + "[GigE] "
00059         self.logdebug = self.ca.logdebugbase + "[GigE] "
00060         logging.info(self.loginfo + "Initializing GigE comms object")
00061         self.mode = 1
00062         self.writeTimeout = 10000
00063         self.readTimeout = 10000
00064         self.payloadsize = (
00065             self.ca.sensor.width
00066             * self.ca.sensor.height
00067             * self.ca.sensor.nframes
00068             * self.ca.sensor.bytesperpixel
00069         )
00070         logging.debug(
00071             self.logdebug + "Payload size: " + str(self.payloadsize) + " bytes"
00072         )
00073         self.skipError = False
00074
00075         self.ZErrorDict = {
00076             0x8000: "Socket Error",
00077             0x8001: "Internal Error",
00078             0x8002: "Illegal Status Code",
00079             0x8003: "Null Parameter",
00080             0x8004: "Out of Memory",
00081             0x8005: "Invalid Connection Type",
00082             0x8006: "Illegal Connection",
00083             0x8007: "Socket Closed Unexpectedly",
00084             0x8008: "Timeout",
00085             0x8009: "Illegal Parameter",
00086         }
00087
00088         if self.ca.port:
00089             logging.debug(
00090                 self.logdebug + "Port supplied to GigE.py: " + str(self.ca.port)
00091             )
00092             if isinstance(self.ca.port, int) and 0 < self.ca.port < 65536:
00093                 self.dport = self.ca.port
00094             else:
00095                 logging.error(
00096                     self.logerr + "Invalid port number supplied, defaulting to "
00097                     "20482 "
00098                 )
00099                 self.dport = 20482
00100         else:
00101             self.dport = 20482 # default
00102
00103         self.ca.port = self.dport
00104         logging.debug(self.logdebug + "Port used by GigE.py: " + str(self.dport))
00105
00106         logging.debug(self.logdebug + "CPU architecture: " + str(self.ca.arch))
00107         if self.ca.arch == "64bit":
00108             arch = "64"
```

```

00109         else:
00110             arch = "32"
00111
00112             logging.debug(self.logdebug + "Operating system: " + str(self.ca.platform))
00113             if self.ca.platform == "Windows":
00114                 lib_name = "ZestETM1.dll"
00115             elif self.ca.platform == "Linux" or self.ca.platform == "Darwin":
00116                 lib_name = "libZestETM1.so"
00117             else:
00118                 logging.warning(
00119                     self.logwarn + "System does not self-identify as Linux, Windows, "
00120                     "or Mac. Assuming posix-style libraries "
00121                 )
00122                 lib_name = "libZestETM1.so"
00123
00124             self.closecard = False
00125
00126             libpath = os.path.join(self.ca.packageroot, "comms", "ZestETM1", arch, lib_name)
00127             self._zest = C.DLL(libpath)
00128
00129             self.CardInfo = self.ZESTETM1_CARD_INFO()
00130             self.CardInfoP = C.pointer(self.CardInfo)
00131
00132             # functions
00133             self.ZCountCards = self._zest.ZestETM1CountCards
00134             self.ZCountCards.argtypes = [
00135                 C.POINTER(C.c_ulong),
00136                 C.POINTER(C.POINTER(self.ZESTETM1_CARD_INFO)),
00137                 C.c_int,
00138             ]
00139
00140             self.ZOpenConnection = self._zest.ZestETM1OpenConnection
00141             self.ZOpenConnection.argtypes = [
00142                 C.POINTER(self.ZESTETM1_CARD_INFO),
00143                 C.c_int,
00144                 C.c_ushort,
00145                 C.c_ushort,
00146                 C.POINTER(C.c_void_p),
00147             ]
00148
00149             self.ZWriteData = self._zest.ZestETM1WriteData
00150             self.ZWriteData.argtypes = [
00151                 C.c_void_p,
00152                 C.c_void_p,
00153                 C.c_ulong,
00154                 C.POINTER(C.c_ulong),
00155                 C.c_ulong,
00156             ]
00157
00158             self.ZReadData = self._zest.ZestETM1ReadData
00159             self.ZReadData.argtypes = [
00160                 C.c_void_p,
00161                 C.c_void_p,
00162                 C.c_ulong,
00163                 C.POINTER(C.c_ulong),
00164                 C.c_ulong,
00165             ]
00166
00167             self.Connection = C.c_void_p()
00168             self.openDevice(self.ca.timeout)
00169

```

6.4.3 Member Function Documentation

6.4.3.1 arm()

```

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

```

Puts camera into wait state for trigger. Mode determines source; defaults to

'Hardware'

Args:

mode: 'Software'|'S' activates software, disables hardware triggering
 'Hardware'|'H' activates hardware, disables software triggering
 Hardware is the default

Returns:

tuple (error, response string)

Definition at line 221 of file [GigE.py](#).

```
00221     def arm(self, mode):
00222         """
00223         Puts camera into wait state for trigger. Mode determines source; defaults to
00224         'Hardware'
00225
00226         Args:
00227             mode: 'Software'|'S' activates software, disables hardware triggering
00228                   'Hardware'|'H' activates hardware, disables software triggering
00229                   Hardware is the default
00230
00231         Returns:
00232             tuple (error, response string)
00233         """
00234         if not mode:
00235             mode = "Hardware"
00236             logging.info(self.loginfo + "arm")
00237             logging.debug(self.logdebug + "arming mode: " + str(mode))
00238             self.ca.clearStatus()
00239             self.ca.latchPots()
00240             err, resp = self.ca.startCapture(mode)
00241             if err:
00242                 logging.error(self.logerr + "unable to arm camera")
00243             else:
00244                 self.ca.armed = True
00245                 self.skipError = True
00246             return err, resp
00247
```

6.4.3.2 closeDevice()

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

Close connection to Orange Tree card and free resources

Definition at line 467 of file [GigE.py](#).

```
00467     def closeDevice(self):
00468         """
00469         Close connection to Orange Tree card and free resources
00470         """
00471         logging.debug(self.logdebug + "Closing connection to Orange Tree card")
00472         self._zest.ZestETM1CloseConnection(self.Connection)
00473         if self.closecard:
00474             try:
00475                 self._zest.ZestETM1FreeCards(self.CardInfoP)
00476             except SystemExit:
00477                 raise
00478             except KeyboardInterrupt:
00479                 raise
00480             except Exception:
00481                 logging.error(self.logerr + "Error reported in OT card closure")
00482         self._zest.ZestETM1Close()
00483
```

6.4.3.3 getCardInfo()

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

Prints status message with information returned by OT card

Definition at line 493 of file [GigE.py](#).

```
00493     def getCardInfo(self):
00494         """
00495         Prints status message with information returned by OT card
00496         """
00497         ci = self.CardInfoP.contents
00498         print("GigE Card Status:")
00499         print("-----")
00500         print("IP: " + ".".join(str(e) for e in [b for b in ci.IPAddr]))
00501         print("ControlPort: " + str(ci.ControlPort))
00502         print("Timeout: " + str(ci.Timeout))
00503         print("HTTPPort: " + str(ci.HTTPPort))
00504         print("MACAddr: " + ".".join(format(e, "02X") for e in [b for b in ci.MACAddr]))
00505         print("SubNet: " + ".".join(str(e) for e in [b for b in ci.SubNet]))
00506         print("Gateway: " + ".".join(str(e) for e in [b for b in ci.Gateway]))
00507         print("SerialNumber: " + str(ci.SerialNumber))
00508         print("FirmwareVersion: " + str(ci.FirmwareVersion))
00509         print("HardwareVersion: " + str(ci.HardwareVersion))
00510         print("-----")
00511
```

6.4.3.4 getCardIP()

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

Query IP address of OT card

Returns: address of OT card as list of bytes

Definition at line 484 of file [GigE.py](#).

```
00484     def getCardIP(self):
00485         """
00486         Query IP address of OT card
00487
00488         Returns: address of OT card as list of bytes
00489         """
00490         return self.CardInfo.IPAddr
00491
```

6.4.3.5 openDevice()

```
nsCamera.comms.GigE.GigE.openDevice (
    self,
    timeout = 30 )
```

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

Args:

timeout: timeout in seconds for attempting to connect to a card

Definition at line 402 of file [GigE.py](#).

```

00402     def openDevice(self, timeout=30):
00403         """
00404         Find Orange Tree card and open a connection; if IP is supplied as parameter for
00405         the CameraAssembler, bypass network search and connect directly to indicated
00406         IP address
00407
00408         Args:
00409             timeout: timeout in seconds for attempting to connect to a card
00410         """
00411         err = self._zest.ZestETM1Init()
00412         if err:
00413             logging.critical(self.logcrit + "ZestETM1Init failure")
00414             sys.exit(1)
00415         logging.info(self.loginfo + "searching for Orange Tree cards")
00416         NumCards = C.c_ulong(0)
00417
00418         if self.ca.iplist:
00419             ubyte4 = C.c_ubyte * 4
00420             self.CardInfo.IPAddr = ubyte4(*self.ca.iplist)
00421             self.CardInfo.ControlPort = C.c_ushort(self.dport)
00422             self.CardInfo.Timeout = C.c_ulong(self.writeTimeout)
00423             self.closecard = False
00424         else:
00425             wait = 0
00426             while True:
00427                 logging.debug(
00428                     self.logdebug + "openDevice: connection wait = " + str(wait)
00429                 )
00430                 err = self.ZCountCards(C.byref(NumCards), C.byref(self.CardInfoP), 1000)
00431                 if err:
00432                     logging.critical(self.logcrit + "CountCards failure")
00433                     sys.exit(1)
00434                 if NumCards.value > 0:
00435                     break
00436                 if wait == timeout:
00437                     logging.critical(
00438                         "{}No Orange Tree cards found in {} seconds".format(
00439                             self.logcrit, timeout
00440                         )
00441                     )
00442                     sys.exit(1)
00443                 elif not wait % 5:
00444                     logging.info(
00445                         "{}Still trying to connect after {} seconds...".format(
00446                             self.loginfo, wait
00447                         )
00448                     )
00449                 wait += 1
00450             logging.info(
00451                 self.loginfo + str(NumCards.value) + " Orange Tree card(s) found"
00452             ) # TODO: add check for GigE bit in board description
00453         err = self.ZOpenConnection(
00454             self.CardInfoP, 0, self.dport, 0, C.byref(self.Connection)
00455         )
00456         if err:
00457             if err == 0x4000:
00458                 logging.warning(
00459                     self.logerr + "OT Card emitted an undefined warning message"
00460                 )
00461             else:
00462                 logging.critical(
00463                     self.logcrit + "OpenConnection failure: " + self.ZErrorDict[err]
00464                 )
00465             sys.exit(1)
00466

```

6.4.3.6 readFrames()

```

nsCamera.comms.GigE.GigE.readFrames (
    self,
    waitOnSRAM,
    timeout = 0,
    fast = False,
    columns = 1 )

```


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
 columns: 1 for single image per frame, 2 for separate hemisphere images

Returns:

list of numpy arrays OR raw text stream

Definition at line 248 of file [GigE.py](#).

```
00248     def readFrames(self, waitOnSRAM, timeout=0, fast=False, columns=1):
00249         """
00250         Copies image data from board into numpy arrays.
00251
00252         Args:
00253             waitOnSRAM: if True, wait until SRAM_READY flag is asserted to begin copying
00254             data
00255             timeout: passed to waitForSRAM; after this many seconds begin copying data
00256             irrespective of SRAM_READY status; 'zero' means wait indefinitely
00257             WARNING: If acquisition fails, the SRAM will not contain a current image,
00258             but the code will copy the data anyway
00259             fast: if False, parse and convert frames to numpy arrays; if True, return
00260             unprocessed text stream
00261             columns: 1 for single image per frame, 2 for separate hemisphere images
00262
00263         Returns:
00264             list of numpy arrays OR raw text stream
00265
00266         """
00267         frames, _, _ = self.readoff(waitOnSRAM, timeout, fast, columns)
00268         return frames
00269
```

6.4.3.7 readoff()

```
nsCamera.comms.GigE.GigE.readoff (
    self,
    waitOnSRAM,
    timeout = 0,
    fast = False,
    columns = 1 )
```

Copies image data from board into numpy arrays; returns data, length of data, and error messages. Use 'readFrames()' unless you require this additional information

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
 columns: 1 for single image per frame, 2 for separate hemisphere images

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 270 of file [GigE.py](#).

```

00270     def readoff(self, waitOnSRAM, timeout=0, fast=False, columns=1):
00271         """
00272         Copies image data from board into numpy arrays; returns data, length of data,
00273         and error messages. Use 'readFrames()' unless you require this additional
00274         information
00275
00276         Args:
00277             waitOnSRAM: if True, wait until SRAM_READY flag is asserted to begin copying
00278             data
00279             timeout: passed to waitForSRAM; after this many seconds begin copying data
00280             irrespective of SRAM_READY status; 'zero' means wait indefinitely
00281             WARNING: If acquisition fails, the SRAM will not contain a current image,
00282             but the code will copy the data anyway
00283             fast: if False, parse and convert frames to numpy arrays; if True, return
00284             unprocessed text stream
00285             columns: 1 for single image per frame, 2 for separate hemisphere images
00286
00287         Returns:
00288             tuple (list of numpy arrays OR raw text stream, length of downloaded payload
00289             in bytes, payload error flag) since CRC check is handled by TCP/IP,
00290             payload error flag is always False for GigE
00291         """
00292         logging.info(self.loginfo + "readoff")
00293         logging.debug(
00294             self.logdebug
00295             + "readoff: waitonSRAM = "
00296             + str(waitOnSRAM)
00297             + "; timeout = "
00298             + str(timeout)
00299             + "; fast = "
00300             + str(fast)
00301         )
00302         # Wait for data to be ready on board
00303         # Skip wait only if explicitly tagged 'False' ('None' defaults to True)
00304         if waitOnSRAM is not False:
00305             self.ca.waitForSRAM(timeout)
00306         self.skipError = False
00307         err, rval = self.ca.readSRAM()
00308         if err:
00309             logging.error(self.logerr + "Error detected in readSRAM")
00310         elif self.ca.boardname == "llnl_v4":
00311             # self.ca.setSubregister('SWACK','1')
00312             pass
00313         # extract the data. Remove header; the FPGA returns a packet without the CRC
00314         # suffix
00315         logging.debug(self.logdebug + "readoff: first 64 chars: " + str(rval[0:64]))
00316         data = rval[32:]
00317         if fast:
00318             return data, len(data) // 2, bool(err)
00319         else:
00320             parsed = generateFrames(self.ca, data, columns)
00321             return parsed, len(data) // 2, bool(err)
00322

```

6.4.3.8 readSerial()

```

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 363 of file [GigE.py](#).

```
00363     def readSerial(self, size, timeout=None):
00364         """
00365         Read bytes from the serial port. Does not verify packets.
00366
00367         Args:
00368             size: number of bytes to read
00369             timeout: serial timeout in sec (defaults to self.readTimeout)
00370
00371         Returns:
00372             tuple (error string, string read from serial port)
00373         """
00374         logging.debug(
00375             self.logdebug
00376             + "readSerial: size = "
00377             + str(size)
00378             + "; timeout = "
00379             + str(timeout)
00380         )
00381         if not timeout:
00382             timeout = self.readTimeout
00383         inbuff = C.create_string_buffer(size + 1)
00384         inbuffp = C.pointer(inbuff)
00385         readlen = C.c_ulong(0)
00386         err = self.ZReadData(self.Connection, inbuffp, size, C.byref(readlen), timeout)
00387         if err:
00388             if self.skipError:
00389                 logging.debug(
00390                     self.logdebug + "readSerial: skipped error: " + self.ZErrorDict[err]
00391                 )
00392                 self.skipError = False
00393             elif err == 0x4000:
00394                 logging.warning(
00395                     self.logerr + "OT Card emitted an undefined warning message"
00396                 )
00397             else:
00398                 logging.error(self.logerr + "readSerial error: " + self.ZErrorDict[err])
00399         return bytes2str(inbuff.raw)[:readlen]
00400
```

6.4.3.9 sendCMD()

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

Submit packet and verify the 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 170 of file [GigE.py](#).

```
00170     def sendCMD(self, pkt):
00171         """
00172         Submit packet and verify the response packet.
00173         Packet communications with FPGA omit CRC suffix, so adds fake CRC bytes to
```

```

00174         response
00175
00176     Args:
00177         pkt: Packet object
00178
00179     Returns:
00180         tuple (error, response string)
00181     """
00182
00183     pktStr = pkt.pktStr()[0:16]
00184     logging.debug(self.logdebug + "sendCMD packet: " + str(pktStr))
00185     err = ""
00186     self.ca.writeSerial(pktStr)
00187     if (
00188         hasattr(self.ca, "board")
00189         and pktStr[4] == "0"
00190         and pktStr[5:8] == self.ca.board.registers["SRAM_CTL"]
00191     ):
00192         bufsize = self.payloadsize + 16
00193         resptext = self.readSerial(bufsize)
00194
00195         if len(resptext) < 32:
00196             logging.debug(self.logdebug + "sendCMD resptext = " + str(resptext))
00197         else:
00198             logging.debug(
00199                 self.logdebug
00200                 + "sendCMD resptext (truncated) = "
00201                 + str(resptext)[0:32]
00202             )
00203
00204         if len(resptext) < bufsize + 16:
00205             err += (
00206                 self.logerr + "sendCMD- packet too small, payload may be incomplete"
00207             )
00208             logging.error(err)
00209         else:
00210             # workaround for initial setup before board object has been initialized
00211             resp = self.readSerial(8)
00212             logging.debug(self.logdebug + "sendCMD response: " + str(resp))
00213             if len(resp) < 8:
00214                 err += self.logerr + "sendCMD- response too small, returning zeros"
00215                 resptext = "00000000000000000000"
00216                 logging.error(err)
00217             else:
00218                 resptext = resp + "0000"
00219     return err, resptext
00220

```

6.4.3.10 writeSerial()

```

nsCamera.comms.GigE.GigE.writeSerial (
    self,
    outstring,
    timeout )

```

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 323 of file [GigE.py](#).

```

00323     def writeSerial(self, outstring, timeout):
00324         """
00325         Transmit string to board
00326
00327         Args:

```

```

00328         outstring: string to write
00329         timeout: serial timeout in sec (defaults to self.writeTimeout)
00330
00331     Returns:
00332         integer number of bytes written
00333     """
00334     logging.debug(
00335         self.logdebug
00336         + "writeSerial: outstring = "
00337         + str(outstring)
00338         + "; timeout = "
00339         + str(timeout)
00340     )
00341     if not timeout:
00342         timeout = self.writeTimeout
00343     outstring = str2bytes(outstring)
00344     outbuff = C.create_string_buffer(outstring)
00345     outbuffp = C.pointer(outbuff)
00346     outbufflen = len(outstring)
00347     writelen = C.c_ulong(0)
00348     err = self.ZWriteData(
00349         self.Connection, outbuffp, outbufflen, C.byref(writelen), timeout
00350     )
00351     if err:
00352         if err == 0x4000:
00353             logging.warning(
00354                 self.logerr + "OT Card emitted an undefined warning message"
00355             )
00356         else:
00357             logging.error(
00358                 self.logerr + "writeSerial error: " + self.ZErrorDict[err]
00359             )
00360     logging.debug(self.logdebug + "writeSerial: writelen = " + str(writelen))
00361     return writelen
00362

```

6.4.4 Member Data Documentation

6.4.4.1 `_zest`

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

Definition at line 127 of file [GigE.py](#).

6.4.4.2 `ca`

`nsCamera.comms.GigE.GigE.ca`

Definition at line 54 of file [GigE.py](#).

6.4.4.3 `CardInfo`

`nsCamera.comms.GigE.GigE.CardInfo`

Definition at line 129 of file [GigE.py](#).

6.4.4.4 CardInfoP

`nsCamera.comms.GigE.GigE.CardInfoP`

Definition at line 130 of file [GigE.py](#).

6.4.4.5 closecard

`nsCamera.comms.GigE.GigE.closecard`

Definition at line 124 of file [GigE.py](#).

6.4.4.6 Connection

`nsCamera.comms.GigE.GigE.Connection`

Definition at line 167 of file [GigE.py](#).

6.4.4.7 dport

`nsCamera.comms.GigE.GigE.dport`

Definition at line 93 of file [GigE.py](#).

6.4.4.8 logcrit

`nsCamera.comms.GigE.GigE.logcrit`

Definition at line 55 of file [GigE.py](#).

6.4.4.9 logdebug

`nsCamera.comms.GigE.GigE.logdebug`

Definition at line 59 of file [GigE.py](#).

6.4.4.10 logerr

`nsCamera.comms.GigE.GigE.logerr`

Definition at line 56 of file [GigE.py](#).

6.4.4.11 loginfo

`nsCamera.comms.GigE.GigE.loginfo`

Definition at line 58 of file [GigE.py](#).

6.4.4.12 logwarn

`nsCamera.comms.GigE.GigE.logwarn`

Definition at line 57 of file [GigE.py](#).

6.4.4.13 mode

`nsCamera.comms.GigE.GigE.mode`

Definition at line 61 of file [GigE.py](#).

6.4.4.14 payloadsize

`nsCamera.comms.GigE.GigE.payloadsize`

Definition at line 64 of file [GigE.py](#).

6.4.4.15 readTimeout

`nsCamera.comms.GigE.GigE.readTimeout`

Definition at line 63 of file [GigE.py](#).

6.4.4.16 skipError

`nsCamera.comms.GigE.GigE.skipError`

Definition at line 73 of file [GigE.py](#).

6.4.4.17 writeTimeout

`nsCamera.comms.GigE.GigE.writeTimeout`

Definition at line 62 of file [GigE.py](#).

6.4.4.18 ZCountCards

`nsCamera.comms.GigE.GigE.ZCountCards`

Definition at line 133 of file [GigE.py](#).

6.4.4.19 ZErrorDict

`nsCamera.comms.GigE.GigE.ZErrorDict`

Definition at line 75 of file [GigE.py](#).

6.4.4.20 ZESTETM1_CARD_INFO

`nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO`

Definition at line 136 of file [GigE.py](#).

6.4.4.21 ZOpenConnection

`nsCamera.comms.GigE.GigE.ZOpenConnection`

Definition at line 140 of file [GigE.py](#).

6.4.4.22 ZReadData

`nsCamera.comms.GigE.GigE.ZReadData`

Definition at line 158 of file [GigE.py](#).

6.4.4.23 ZWriteData

`nsCamera.comms.GigE.GigE.ZWriteData`

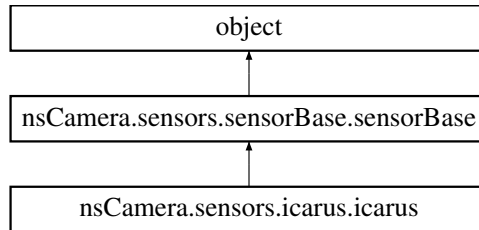
Definition at line 149 of file [GigE.py](#).

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

- [C:/Users/hill35/git/camera_python/nsCamera/comms/GigE.py](#)

6.5 nsCamera.sensors.icarus.icarus Class Reference

Inheritance diagram for nsCamera.sensors.icarus.icarus:



Public Member Functions

- [__init__](#) (self, ca)
- [checkSensorVoltStat](#) (self)
- [sensorSpecific](#) (self)

Public Member Functions inherited from [nsCamera.sensors.sensorBase.sensorBase](#)

- [init_board_specific](#) (self)
- [setInterlacing](#) (self, ifactor)
- [setHighFullWell](#) (self, flag)
- [setZeroDeadTime](#) (self, flag)
- [setTriggerDelay](#) (self, delay)
- [setPhiDelay](#) (self, delay)
- [setExtClk](#) (self, delay)
- [setTiming](#) (self, side="AB", sequence=None, delay=0)
- [setArbTiming](#) (self, side="AB", sequence=None)
- [getTiming](#) (self, side, actual)
- [setManualShutters](#) (self, timing=None)
- [setManualTiming](#) (self, timing=None)
- [getManualTiming](#) (self)
- [getSensTemp](#) (self, scale=None, offset=None, slope=None, dec=None)
- [selectOscillator](#) (self, osc=None)
- [parseReadoff](#) (self, frames, columns)
- [getSensorStatus](#) (self)
- [reportStatusSensor](#) (self, statusbits, statusbits2)

Public Attributes

- [ca](#)
- [sens_registers](#)
- [sens_subregisters](#)

Public Attributes inherited from [nsCamera.sensors.sensorBase.sensorBase](#)

- [ca](#)
- [logcrit](#)
- [logerr](#)
- [logwarn](#)
- [loginfo](#)
- [logdebug](#)
- [fpganumID](#)
- [sensfam](#)

Static Public Attributes

- str [specwarn](#) = " and the use of the Icarus model 1 sensor"
- int [minframe](#) = 1
- int [maxframe](#) = 2
- int [maxwidth](#) = 512
- int [maxheight](#) = 1024
- int [bytesperpixel](#) = 2
- int [icarustype](#) = 1
- int [fpganumID](#) = 1
- str [detect](#) = "ICARUS_DET"
- str [sensfam](#) = "Icarus"
- str [loglabel](#) = "[Icarus1] "
- int [firstframe](#) = 1
- int [lastframe](#) = 2
- int [nframes](#) = 2
- int [width](#) = 512
- int [height](#) = 1024
- int [firstrow](#) = 0
- int [lastrow](#) = 1023
- list [interlacing](#) = [0, 0]
- int [columns](#) = 1
- bool [padToFull](#) = True

6.5.1 Detailed Description

Definition at line 27 of file [icarus.py](#).

6.5.2 Constructor & Destructor Documentation

6.5.2.1 __init__()

```
nsCamera.sensors.icarus.icarus.__init__ (
    self,
    ca )
```

Reimplemented from [nsCamera.sensors.sensorBase.sensorBase](#).

Definition at line 53 of file [icarus.py](#).

```
00053     def __init__(self, ca):
00054         self.ca = ca
00055         super(icarus, self).__init__(ca)
00056
00057         self.sens_registers = OrderedDict(
00058             {
00059                 "VRESET_WAIT_TIME": "03E",
00060                 "ICARUS_VER_SEL": "041",
00061                 "VRESET_HIGH_VALUE": "04A",
00062                 "MISC_SENSOR_CTL": "04C",
00063                 "MANUAL_SHUTTERS_MODE": "050",
00064                 "W0_INTEGRATION": "051",
00065                 "W0_INTERFRAME": "052",
00066                 "W1_INTEGRATION": "053",
00067                 "W1_INTERFRAME": "054",
00068                 "W2_INTEGRATION": "055",
00069                 "W2_INTERFRAME": "056",
00070                 "W3_INTEGRATION": "057",
00071                 "W0_INTEGRATION_B": "058",
00072                 "W0_INTERFRAME_B": "059",
00073                 "W1_INTEGRATION_B": "05A",
00074                 "W1_INTERFRAME_B": "05B",
00075                 "W2_INTEGRATION_B": "05C",
00076                 "W2_INTERFRAME_B": "05D",
00077                 "W3_INTEGRATION_B": "05E",
00078                 "TIME_ROW_DCD": "05F",
00079             }
00080         )
00081
00082         self.sens_subregisters = [
00083
00084             ("MANSHUT_MODE", "MANUAL_SHUTTERS_MODE", 0, 1, True),
00085             ("REVREAD", "CTRL_REG", 4, 1, True),
00086             ("PDBIAS_LOW", "CTRL_REG", 6, 1, True),
00087             ("ROWDCD_CTL", "CTRL_REG", 7, 1, True),
00088             ("ACCUMULATION_CTL", "MISC_SENSOR_CTL", 0, 1, True),
00089             ("HST_TST_ANRST_EN", "MISC_SENSOR_CTL", 1, 1, True),
00090             ("HST_TST_BNRST_EN", "MISC_SENSOR_CTL", 2, 1, True),
00091             ("HST_TST_ANRST_IN", "MISC_SENSOR_CTL", 3, 1, True),
00092             ("HST_TST_BNRST_IN", "MISC_SENSOR_CTL", 4, 1, True),
00093             ("HST_PXL_RST_EN", "MISC_SENSOR_CTL", 5, 1, True),
00094             ("HST_CONT_MODE", "MISC_SENSOR_CTL", 6, 1, True),
00095             ("COL_DCD_EN", "MISC_SENSOR_CTL", 7, 1, True),
00096             ("COL_READOUT_EN", "MISC_SENSOR_CTL", 8, 1, True),
00097             ("READOFF_DELAY_EN", "TRIGGER_CTL", 4, 1, True),
00098
00099             ("STAT_W3TOPAEDGE1", "STAT_REG", 3, 1, False),
00100             ("STAT_W3TOPBEDGE1", "STAT_REG", 4, 1, False),
00101             ("STAT_HST_ALL_W_EN_DETECTED", "STAT_REG", 12, 1, False),
00102             ("PDBIAS_UNREADY", "STAT_REG2", 5, 1, False),
00103         ]
00104
00105         if self.ca.boardname == "l1n1_v1":
00106             self.sens_subregisters.append(
00107                 ("VRESET_HIGH", "VRESET_HIGH_VALUE", 7, 8, True)
00108             )
00109         else:
00110             self.sens_subregisters.extend(
00111                 [
00112                     ("VRESET_HIGH", "VRESET_HIGH_VALUE", 15, 16, True),
00113                     ("READOFF_DELAY_EN", "TRIGGER_CTL", 4, 1, True),
00114                 ]
00115             )
00116         self.sens_registers.update({"DELAY_ASSERTION_ROWDCD_EN": "04F"})
00117
00118     00119
00120     00120
```

6.5.3 Member Function Documentation

6.5.3.1 checkSensorVoltStat()

```
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

Reimplemented from [nsCamera.sensors.sensorBase.sensorBase](#).

Definition at line 121 of file [icarus.py](#).

```
00121     def checkSensorVoltStat(self):
00122         """
00123         Checks register tied to sensor select jumpers to confirm match with sensor
00124         object
00125
00126         Returns:
00127             boolean, True if jumpers select for Icarus sensor
00128         """
00129         logging.debug(self.logdebug + "checkSensorVoltStat")
00130         err, status = self.ca.getSubregister("ICARUS_DET")
00131         if err:
00132             logging.error(self.logerr + "unable to confirm sensor status")
00133             return False
00134         if not int(status):
00135             logging.error(self.logerr + "Icarus sensor not detected")
00136             return False
00137         return True
00138
```

6.5.3.2 sensorSpecific()

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

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

Definition at line 139 of file [icarus.py](#).

```
00139     def sensorSpecific(self):
00140         """
00141         Returns:
00142             list of tuples, (Sensor-specific register, default setting)
00143         """
00144         icarussettings = [
00145             ("ICARUS_VER_SEL", "00000001"),
00146             ("FPA_FRAME_INITIAL", "00000001"),
00147             ("FPA_FRAME_FINAL", "00000002"),
00148             ("FPA_ROW_INITIAL", "00000000"),
00149             ("FPA_ROW_FINAL", "000003FF"),
00150             ("VRESET_WAIT_TIME", "000927C0"),
00151             ("HS_TIMING_DATA_BHI", "00000000"),
00152             ("HS_TIMING_DATA_BLO", "00006666"), # 0db6 = 2-1; 6666 = 2-2
00153             ("HS_TIMING_DATA_AHI", "00000000"),
00154             ("HS_TIMING_DATA_ALO", "00006666"),
00155         ]
00156         if self.ca.boardname == "1ln1_v1":
```

```

00157         icarussettings.append(
00158             ("VRESET_HIGH_VALUE", "000000D5") # 3.3 V (FF = 3.96)
00159         )
00160     else:
00161         icarussettings.append(("VRESET_HIGH_VALUE", "0000FFFF"))
00162     return icarussettings
00163
00164
00165 """
00166 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00167 LLNL-CODE-838080
00168
00169 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00170 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00171 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00172 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00173 be made under this license.
00174 """

```

6.5.4 Member Data Documentation

6.5.4.1 bytesperpixel

```
int nsCamera.sensors.icarus.icarus.bytesperpixel = 2 [static]
```

Definition at line 36 of file [icarus.py](#).

6.5.4.2 ca

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

Definition at line 54 of file [icarus.py](#).

6.5.4.3 columns

```
int nsCamera.sensors.icarus.icarus.columns = 1 [static]
```

Definition at line 50 of file [icarus.py](#).

6.5.4.4 detect

```
str nsCamera.sensors.icarus.icarus.detect = "ICARUS_DET" [static]
```

Definition at line 39 of file [icarus.py](#).

6.5.4.5 firstframe

```
int nsCamera.sensors.icarus.icarus.firstframe = 1 [static]
```

Definition at line 42 of file [icarus.py](#).

6.5.4.6 firstrow

```
int nsCamera.sensors.icarus.icarus.firstrow = 0 [static]
```

Definition at line 47 of file [icarus.py](#).

6.5.4.7 fpganumID

```
int nsCamera.sensors.icarus.icarus.fpganumID = 1 [static]
```

Definition at line 38 of file [icarus.py](#).

6.5.4.8 height

```
int nsCamera.sensors.icarus.icarus.height = 1024 [static]
```

Definition at line 46 of file [icarus.py](#).

6.5.4.9 icarustype

```
int nsCamera.sensors.icarus.icarus.icarustype = 1 [static]
```

Definition at line 37 of file [icarus.py](#).

6.5.4.10 interlacing

```
list nsCamera.sensors.icarus.icarus.interlacing = [0, 0] [static]
```

Definition at line 49 of file [icarus.py](#).

6.5.4.11 lastframe

```
int nsCamera.sensors.icarus.icarus.lastframe = 2 [static]
```

Definition at line 43 of file [icarus.py](#).

6.5.4.12 lastrow

```
int nsCamera.sensors.icarus.icarus.lastrow = 1023 [static]
```

Definition at line 48 of file [icarus.py](#).

6.5.4.13 loglabel

```
str nsCamera.sensors.icarus.icarus.loglabel = "[Icarus1] " [static]
```

Definition at line 41 of file [icarus.py](#).

6.5.4.14 maxframe

```
int nsCamera.sensors.icarus.icarus.maxframe = 2 [static]
```

Definition at line 30 of file [icarus.py](#).

6.5.4.15 maxheight

```
int nsCamera.sensors.icarus.icarus.maxheight = 1024 [static]
```

Definition at line 35 of file [icarus.py](#).

6.5.4.16 maxwidth

```
int nsCamera.sensors.icarus.icarus.maxwidth = 512 [static]
```

Definition at line 34 of file [icarus.py](#).

6.5.4.17 minframe

```
int nsCamera.sensors.icarus.icarus.minframe = 1 [static]
```

Definition at line 29 of file [icarus.py](#).

6.5.4.18 nframes

```
int nsCamera.sensors.icarus.icarus.nframes = 2 [static]
```

Definition at line 44 of file [icarus.py](#).

6.5.4.19 padToFull

```
bool nsCamera.sensors.icarus.icarus.padToFull = True [static]
```

Definition at line 51 of file [icarus.py](#).

6.5.4.20 sens_registers

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

Definition at line 57 of file [icarus.py](#).

6.5.4.21 sens_subregisters

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

Definition at line 82 of file [icarus.py](#).

6.5.4.22 sensfam

```
str nsCamera.sensors.icarus.icarus.sensfam = "Icarus" [static]
```

Definition at line 40 of file [icarus.py](#).

6.5.4.23 specwarn

```
str nsCamera.sensors.icarus.icarus.specwarn = " and the use of the Icarus model 1 sensor" [static]
```

Definition at line 28 of file [icarus.py](#).

6.5.4.24 width

```
int nsCamera.sensors.icarus.icarus.width = 512 [static]
```

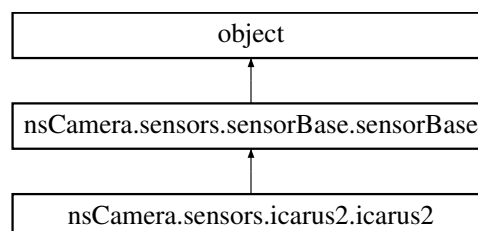
Definition at line 45 of file [icarus.py](#).

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

- C:/Users/hill35/git/camera_python/nsCamera/sensors/[icarus.py](#)

6.6 nsCamera.sensors.icarus2.icarus2 Class Reference

Inheritance diagram for nsCamera.sensors.icarus2.icarus2:



Public Member Functions

- [__init__](#) (self, [ca](#))
- [sensorSpecific](#) (self)

Public Member Functions inherited from [nsCamera.sensors.sensorBase.sensorBase](#)

- [init_board_specific](#) (self)
- [checkSensorVoltStat](#) (self)
- [setInterlacing](#) (self, ifactor)
- [setHighFullWell](#) (self, flag)
- [setZeroDeadTime](#) (self, flag)
- [setTriggerDelay](#) (self, delay)
- [setPhiDelay](#) (self, delay)
- [setExtClk](#) (self, delay)
- [setTiming](#) (self, side="AB", sequence=None, delay=0)
- [setArbTiming](#) (self, side="AB", sequence=None)
- [getTiming](#) (self, side, actual)
- [setManualShutters](#) (self, timing=None)
- [setManualTiming](#) (self, timing=None)
- [getManualTiming](#) (self)
- [getSensTemp](#) (self, scale=None, offset=None, slope=None, dec=None)
- [selectOscillator](#) (self, osc=None)
- [parseReadoff](#) (self, frames, columns)
- [getSensorStatus](#) (self)
- [reportStatusSensor](#) (self, statusbits, statusbits2)

Public Attributes

- [ca](#)
- [sens_registers](#)
- [sens_subregisters](#)

Public Attributes inherited from [nsCamera.sensors.sensorBase.sensorBase](#)

- [ca](#)
- [logcrit](#)
- [logerr](#)
- [logwarn](#)
- [loginfo](#)
- [logdebug](#)
- [fpganumID](#)
- [sensfam](#)

Static Public Attributes

- str `specwarn` = ""
- int `minframe` = 0
- int `maxframe` = 3
- int `maxwidth` = 512
- int `maxheight` = 1024
- int `bytesperpixel` = 2
- int `icarustype` = 0
- int `fpganumID` = 1
- str `detect` = "ICARUS_DET"
- str `sensfam` = "Icarus"
- str `loglabel` = "[Icarus2] "
- int `firstframe` = 0
- int `lastframe` = 3
- int `nframes` = 4
- int `width` = 512
- int `height` = 1024
- int `firstrow` = 0
- int `lastrow` = 1023
- list `interlacing` = [0, 0]
- int `columns` = 1
- bool `padToFull` = True

6.6.1 Detailed Description

Definition at line 24 of file `icarus2.py`.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 `__init__()`

```
nsCamera.sensors.icarus2.icarus2.__init__ (
    self,
    ca )
```

Reimplemented from `nsCamera.sensors.sensorBase.sensorBase`.

Definition at line 47 of file `icarus2.py`.

```
00047     def __init__(self, ca):
00048         self.ca = ca
00049         super(icarus2, self).__init__(ca)
00050
00051         self.sens_registers = OrderedDict(
00052             {
00053                 "VRESET_WAIT_TIME": "03E",
00054                 "ICARUS_VER_SEL": "041",
00055                 "MISC_SENSOR_CTL": "04C",
00056                 "MANUAL_SHUTTERS_MODE": "050",
00057                 "W0_INTEGRATION": "051",
00058                 "W0_INTERFRAME": "052",
00059                 "W1_INTEGRATION": "053",
00060                 "W1_INTERFRAME": "054",
```

```

00061         "W2_INTEGRATION": "055",
00062         "W2_INTERFRAME": "056",
00063         "W3_INTEGRATION": "057",
00064         "W0_INTEGRATION_B": "058",
00065         "W0_INTERFRAME_B": "059",
00066         "W1_INTEGRATION_B": "05A",
00067         "W1_INTERFRAME_B": "05B",
00068         "W2_INTEGRATION_B": "05C",
00069         "W2_INTERFRAME_B": "05D",
00070         "W3_INTEGRATION_B": "05E",
00071         "TIME_ROW_DCD": "05F",
00072     }
00073 )
00074
00075 self.sens_subregisters = [
00076
00077     ("MANSHUT_MODE", "MANUAL_SHUTTERS_MODE", 0, 1, True),
00078     ("REVREAD", "CTRL_REG", 4, 1, True),
00079     ("PDBIAS_LOW", "CTRL_REG", 6, 1, True),
00080     ("ROWDCD_CTL", "CTRL_REG", 7, 1, True),
00081     ("ACCUMULATION_CTL", "MISC_SENSOR_CTL", 0, 1, True),
00082     ("HST_TST_ANRST_EN", "MISC_SENSOR_CTL", 1, 1, True),
00083     ("HST_TST_BNRST_EN", "MISC_SENSOR_CTL", 2, 1, True),
00084     ("HST_TST_ANRST_IN", "MISC_SENSOR_CTL", 3, 1, True),
00085     ("HST_TST_BNRST_IN", "MISC_SENSOR_CTL", 4, 1, True),
00086     ("HST_PXL_RST_EN", "MISC_SENSOR_CTL", 5, 1, True),
00087     ("HST_CONT_MODE", "MISC_SENSOR_CTL", 6, 1, True),
00088     ("COL_DCD_EN", "MISC_SENSOR_CTL", 7, 1, True),
00089     ("COL_READOUT_EN", "MISC_SENSOR_CTL", 8, 1, True),
00090
00091
00092     ("STAT_W3TOPAEDGE1", "STAT_REG", 3, 1, False),
00093     ("STAT_W3TOPBEDGE1", "STAT_REG", 4, 1, False),
00094     ("STAT_HST_ALL_W_EN_DETECTED", "STAT_REG", 12, 1, False),
00095     ("PDBIAS_UNREADY", "STAT_REG2", 5, 1, False),
00096 ]
00097
00098 if self.ca.boardname == "l1nl_v4":
00099     self.sens_subregisters.append(
00100         ("READOFF_DELAY_EN", "TRIGGER_CTL", 4, 1, True)
00101     )
00102
00103 self.sens_registers.update({"DELAY_ASSERTION_ROWDCD_EN": "04F"})
00104
00105

```

6.6.3 Member Function Documentation

6.6.3.1 sensorSpecific()

```

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

```

Returns:

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

Definition at line 107 of file [icarus2.py](#).

```

00107 def sensorSpecific(self):
00108     """
00109     Returns:
00110         list of tuples, (Sensor-specific register, default setting)
00111     """
00112     return [
00113         ("ICARUS_VER_SEL", "00000000"),
00114         ("FPA_FRAME_INITIAL", "00000000"),
00115         ("FPA_FRAME_FINAL", "00000003"),
00116         ("FPA_ROW_INITIAL", "00000000"),
00117         ("FPA_ROW_FINAL", "000003FF"),
00118         ("HS_TIMING_DATA_BHI", "00000000"),
00119         ("HS_TIMING_DATA_BLO", "00006666", # 0db = 2-1; 6666 = 2-2
00120         ("HS_TIMING_DATA_AHI", "00000000"),
00121         ("HS_TIMING_DATA_ALO", "00006666"),
00122     ]

```

```
00123
00124
00125 """
00126 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00127 LLNL-CODE-838080
00128
00129 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00130 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00131 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00132 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00133 be made under this license.
00134 """
```

6.6.4 Member Data Documentation

6.6.4.1 bytesperpixel

```
int nsCamera.sensors.icarus2.icarus2.bytesperpixel = 2 [static]
```

Definition at line 30 of file [icarus2.py](#).

6.6.4.2 ca

```
nsCamera.sensors.icarus2.icarus2.ca
```

Definition at line 48 of file [icarus2.py](#).

6.6.4.3 columns

```
int nsCamera.sensors.icarus2.icarus2.columns = 1 [static]
```

Definition at line 44 of file [icarus2.py](#).

6.6.4.4 detect

```
str nsCamera.sensors.icarus2.icarus2.detect = "ICARUS_DET" [static]
```

Definition at line 33 of file [icarus2.py](#).

6.6.4.5 firstframe

```
int nsCamera.sensors.icarus2.icarus2.firstframe = 0 [static]
```

Definition at line 36 of file [icarus2.py](#).

6.6.4.6 firstrow

```
int nsCamera.sensors.icarus2.icarus2.firstrow = 0 [static]
```

Definition at line 41 of file [icarus2.py](#).

6.6.4.7 fpganumID

```
int nsCamera.sensors.icarus2.icarus2.fpganumID = 1 [static]
```

Definition at line 32 of file [icarus2.py](#).

6.6.4.8 height

```
int nsCamera.sensors.icarus2.icarus2.height = 1024 [static]
```

Definition at line 40 of file [icarus2.py](#).

6.6.4.9 icarustype

```
int nsCamera.sensors.icarus2.icarus2.icarustype = 0 [static]
```

Definition at line 31 of file [icarus2.py](#).

6.6.4.10 interlacing

```
list nsCamera.sensors.icarus2.icarus2.interlacing = [0, 0] [static]
```

Definition at line 43 of file [icarus2.py](#).

6.6.4.11 lastframe

```
int nsCamera.sensors.icarus2.icarus2.lastframe = 3 [static]
```

Definition at line 37 of file [icarus2.py](#).

6.6.4.12 lastrow

```
int nsCamera.sensors.icarus2.icarus2.lastrow = 1023 [static]
```

Definition at line 42 of file [icarus2.py](#).

6.6.4.13 loglabel

```
str nsCamera.sensors.icarus2.icarus2.loglabel = "[Icarus2] " [static]
```

Definition at line 35 of file [icarus2.py](#).

6.6.4.14 maxframe

```
int nsCamera.sensors.icarus2.icarus2.maxframe = 3 [static]
```

Definition at line 27 of file [icarus2.py](#).

6.6.4.15 maxheight

```
int nsCamera.sensors.icarus2.icarus2.maxheight = 1024 [static]
```

Definition at line 29 of file [icarus2.py](#).

6.6.4.16 maxwidth

```
int nsCamera.sensors.icarus2.icarus2.maxwidth = 512 [static]
```

Definition at line 28 of file [icarus2.py](#).

6.6.4.17 minframe

```
int nsCamera.sensors.icarus2.icarus2.minframe = 0 [static]
```

Definition at line 26 of file [icarus2.py](#).

6.6.4.18 nframes

```
int nsCamera.sensors.icarus2.icarus2.nframes = 4 [static]
```

Definition at line 38 of file [icarus2.py](#).

6.6.4.19 padToFull

```
bool nsCamera.sensors.icarus2.icarus2.padToFull = True [static]
```

Definition at line 45 of file [icarus2.py](#).

6.6.4.20 sens_registers

```
nsCamera.sensors.icarus2.icarus2.sens_registers
```

Definition at line 51 of file [icarus2.py](#).

6.6.4.21 sens_subregisters

```
nsCamera.sensors.icarus2.icarus2.sens_subregisters
```

Definition at line 75 of file [icarus2.py](#).

6.6.4.22 sensfam

```
str nsCamera.sensors.icarus2.icarus2.sensfam = "Icarus" [static]
```

Definition at line 34 of file [icarus2.py](#).

6.6.4.23 specwarn

```
str nsCamera.sensors.icarus2.icarus2.specwarn = "" [static]
```

Definition at line 25 of file [icarus2.py](#).

6.6.4.24 width

```
int nsCamera.sensors.icarus2.icarus2.width = 512 [static]
```

Definition at line 39 of file [icarus2.py](#).

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

- C:/Users/hill35/git/camera_python/nsCamera/sensors/[icarus2.py](#)

6.7 nsCamera.boards.LLNL_v1.IInI_v1 Class Reference

Public Member Functions

- [__init__](#) (self, camassem)
- [initBoard](#) (self)
- [initPots](#) (self)
- [latchPots](#) (self)
- [initSensor](#) (self)
- [configADCs](#) (self)
- [softReboot](#) (self)
- [disarm](#) (self)
- [startCapture](#) (self, mode="Hardware")
- [readSRAM](#) (self)
- [waitForSRAM](#) (self, timeout)
- [getTimer](#) (self)
- [resetTimer](#) (self)
- [enableLED](#) (self, status)
- [setLED](#) (self, LED, status)
- [setPowerSave](#) (self, status)
- [setPPER](#) (self, pollperiod)
- [getTemp](#) (self, scale=None, offset=None, slope=None)
- [getPressure](#) (self, offset, sensitivity, units)
- [clearStatus](#) (self)
- [checkStatus](#) (self)
- [checkStatus2](#) (self)
- [reportStatus](#) (self)
- [reportEdgeDetects](#) (self)
- [dumpStatus](#) (self)

Public Attributes

- [ca](#)
- [logcrit](#)
- [logerr](#)
- [logwarn](#)
- [loginfo](#)
- [logdebug](#)
- [VREF](#)
- [ADC5_mult](#)
- [ADC5_bipolar](#)
- [rs422_baud](#)
- [rs422_cmd_wait](#)
- [icarus_subreg_aliases](#)
- [icarus_monitor_controls](#)
- [daedalus_subreg_aliases](#)
- [daedalus_monitor_controls](#)
- [subreglist](#)

Static Public Attributes

- [registers](#)
- list [subregisters](#)

6.7.1 Detailed Description

Livermore LLNL v1.0 board

Compatible communication protocols: RS422, GigE
Compatible sensors: icarus, icarus2, daedalus

Definition at line 28 of file [LLNL_v1.py](#).

6.7.2 Constructor & Destructor Documentation

6.7.2.1 `__init__()`

```
nsCamera.boards.LLNL_v1.llnl_v1.__init__ (
    self,
    camassem )
```

Definition at line 192 of file [LLNL_v1.py](#).

```
00192     def __init__(self, camassem):
00193         self.ca = camassem
00194         self.logcrit = self.ca.logcritbase + "[LLNL_v1] "
00195         self.logerr = self.ca.logerrbase + "[LLNL_v1] "
00196         self.logwarn = self.ca.logwarnbase + "[LLNL_v1] "
00197         self.loginfo = self.ca.loginfobase + "[LLNL_v1] "
00198         self.logdebug = self.ca.logdebugbase + "[LLNL_v1] "
00199         logging.info(self.loginfo + "initializing board object")
00200         self.VREF = 2.5 # default
00201         self.ADC5_mult = 2 # i.e., monmax = 2 * VREF
00202         # False => monitor range runs 0 to monmax, True => +/- monmax
00203         self.ADC5_bipolar = True
00204         self.rs422_baud = 921600
00205         self.rs422_cmd_wait = 0.3
00206
00207         fpgaNum_pkt = Packet(cmd="1", addr=self.registers["FPGA_NUM"])
00208         fpgaRev_pkt = Packet(cmd="1", addr=self.registers["FPGA_REV"])
00209
00210         _, _ = self.ca.sendCMD(fpgaNum_pkt) # dummy duplicate call
00211         err, rval = self.ca.sendCMD(fpgaNum_pkt)
00212         self.ca.FPGA_NUM = rval[8:16]
00213
00214         err, rval = self.ca.sendCMD(fpgaRev_pkt)
00215         self.ca.FPGA_VERSION = rval[8:16]
00216
00217         # map channels to signal names for abstraction at the camera assembler level;
00218         # each requires a corresponding entry in 'subregisters'
00219
00220         self.icarus_subreg_aliases = OrderedDict(
00221             {
00222                 "COL_BOT_IBIAS_IN": "POT1",
00223                 "HST_A_PDELAY": "POT2",
00224                 "HST_B_NDELAY": "POT3",
00225                 "HST_RO_IBIAS": "POT4",
00226                 "HST_OSC_VREF_IN": "POT5",
00227                 "HST_B_PDELAY": "POT6",
00228                 "HST_OSC_CTL": "POT7",
00229                 "HST_A_NDELAY": "POT8",
00230                 "COL_TOP_IBIAS_IN": "POT9",
00231                 "HST_OSC_R_BIAS": "POT10",
```

```

00232         "VAB": "POT11",
00233         "HST_RO_NC_IBIAS": "POT12",
00234         "VRST": "POT13",
00235         "MON_HST_A_PDELAY": "MON_CH2",
00236         "MON_HST_B_NDELAY": "MON_CH3",
00237         "MON_HST_RO_IBIAS": "MON_CH4",
00238         "MON_HST_OSC_VREF_IN": "MON_CH5",
00239         "MON_HST_B_PDELAY": "MON_CH6",
00240         "MON_HST_OSC_CTL": "MON_CH7",
00241         "MON_HST_A_NDELAY": "MON_CH8",
00242     }
00243 )
00244 # Read-only; identifies controls corresponding to monitors
00245 self.icarus_monitor_controls = OrderedDict(
00246     {
00247         "MON_CH2": "POT2",
00248         "MON_CH3": "POT3",
00249         "MON_CH4": "POT4",
00250         "MON_CH5": "POT5",
00251         "MON_CH6": "POT6",
00252         "MON_CH7": "POT7",
00253         "MON_CH8": "POT8",
00254         # Note: VRST is not measured across the pot; it will read a voltage
00255         # approximately 1 Volt lower than pot13's actual output
00256         "MON_VRST": "POT13",
00257     }
00258 )
00259
00260 self.daedalus_subreg_aliases = OrderedDict(
00261     {
00262         "HST_OSC_CTL": "POT4",
00263         "HST_RO_NC_IBIAS": "POT5",
00264         "HST_OSC_VREF_IN": "POT6",
00265         "VAB": "POT11",
00266         "MON_TSENSEOUT": "MON_CH2",
00267         "MON_BGREF": "MON_CH3",
00268         "MON_HST_OSC_CTL": "MON_CH4",
00269         "MON_HST_RO_NC_IBIAS": "MON_CH5",
00270         "MON_HST_OSC_VREF_IN": "MON_CH6",
00271         "MON_COL_TST_IN": "MON_CH7",
00272         "MON_HST_OSC_PBIAS_PAD": "MON_CH8",
00273     }
00274 )
00275 # Read-only; identifies controls corresponding to monitors
00276 self.daedalus_monitor_controls = OrderedDict(
00277     {
00278         "MON_CH4": "POT4",
00279         "MON_CH5": "POT5",
00280         "MON_CH6": "POT6",
00281         # Note: VRST is not measured across the pot; it will read a voltage
00282         # lower than pot13's actual output
00283         "MON_VRST": "POT13",
00284     }
00285 )
00286
00287 self.subreglist = []
00288 for s in self.subregisters:
00289     self.subreglist.append(s[0].upper())
00290     sr = SubRegister(
00291         self,
00292         name=s[0].upper(),
00293         register=s[1].upper(),
00294         start_bit=s[2],
00295         width=s[3],
00296         writable=s[4],
00297     )
00298     setattr(self, s[0].upper(), sr)
00299
00300 # set voltage ranges for all pots
00301 for n in range(1, 13):
00302     potname = "POT" + str(n)
00303     potobj = getattr(self, potname)
00304     potobj.minV = 0
00305     potobj.maxV = 3.3
00306     # resolution is approximately .0129 V / LSB
00307     potobj.resolution = (1.0 * potobj.maxV - potobj.minV) / potobj.max_value
00308 self.POT13.minV = 0
00309 self.POT13.maxV = 3.96
00310 # POT13 resolution is approximately .0155 V / LSB
00311 self.POT13.resolution = (
00312     1.0 * self.POT13.maxV - self.POT13.minV

```

```
00313         ) / self.POT13.max_value
00314
```

6.7.3 Member Function Documentation

6.7.3.1 checkStatus()

```
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 729 of file LLNL_v1.py.

```
00729     def checkStatus(self):
00730         """
00731         Check status register, convert to reverse-order bit stream (i.e., bit 0 is
00732         statusbits[0])
00733
00734         Returns:
00735         bit string (no '0b') in reversed order
00736         """
00737         logging.debug(self.logdebug + "checkStatus")
00738         err, rval = self.ca.getRegister("STAT_REG")
00739         if not rval:
00740             logging.error(
00741                 self.logerr + "Unable to check status register (zeroes returned)"
00742             )
00743             rval = "0"
00744             rvalbits = bin(int(rval, 16))[2:].zfill(32)
00745             statusbits = rvalbits[::-1]
00746             return statusbits # TODO: add error handling
00747
```

6.7.3.2 checkStatus2()

```
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 748 of file LLNL_v1.py.

```
00748     def checkStatus2(self):
00749         """
00750         Check second status register, convert to reverse-order bit stream (i.e., bit 0
00751         is statusbits[0])
00752
00753         Returns: bit string (no '0b') in reversed order
00754         """
00755         logging.debug(self.logdebug + "checkStatus2")
00756         err, rval = self.ca.getRegister("STAT_REG2")
00757         if not rval:
00758             logging.error(
00759                 self.logerr + "Unable to check status register 2 (zeroes returned)"
00760             )
00761             rval = "0"
00762             rvalbits = bin(int(rval, 16))[2:].zfill(5)
00763             statusbits = rvalbits[::-1]
00764             return statusbits # TODO: add error handling
00765
```

6.7.3.3 clearStatus()

```
nsCamera.boards.LLNL_v1.llnl_v1.clearStatus (
    self )
```

Check status registers to clear them

Returns:
error string

Definition at line 714 of file LLNL_v1.py.

```
00714     def clearStatus(self):
00715         """
00716         Check status registers to clear them
00717
00718         Returns:
00719             error string
00720         """
00721         logging.debug(self.logdebug + "clearStatus")
00722         err1, rval = self.ca.getRegister("STAT_REG_SRC")
00723         err2, rval = self.ca.getRegister("STAT_REG2_SRC")
00724         err = err1 + err2
00725         if err:
00726             logging.error(self.logerr + "clearStatus failed")
00727         return err
00728
```

6.7.3.4 configADCs()

```
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 437 of file LLNL_v1.py.

```
00437     def configADCs(self):
00438         """
00439         Sets default ADC configuration (does not latch settings)
00440
00441         Returns:
00442             tuple (error string, response string) from final control message
00443         """
00444         logging.info(self.loginfo + "configADCs")
00445
00446         control_messages = [
00447             # just in case ADC_RESET was set (pull all ADCs out # of reset)
00448             ("ADC_RESET", "00000000"),
00449             # workaround for uncertain behavior after previous readoff
00450             ("ADC1_CONFIG_DATA", "FFFFFFF"),
00451             ("ADC2_CONFIG_DATA", "FFFFFFF"),
00452             ("ADC3_CONFIG_DATA", "FFFFFFF"),
00453             ("ADC4_CONFIG_DATA", "FFFFFFF"),
00454             ("ADC_CTL", "FFFFFFF"),
00455             ("ADC1_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
00456             ("ADC2_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
00457             ("ADC3_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
00458             ("ADC4_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
00459             ("ADC5_CONFIG_DATA", "81A883FF"), # int Vref 2.50V
00460         ]
00461         return self.ca.submitMessages(control_messages, " configADCs: ")
00462
```

6.7.3.5 disarm()

```
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 475 of file [LLNL_v1.py](#).

```
00475     def disarm(self):
00476         """
00477         Takes camera out of trigger wait state. Has no effect if camera is not already
00478         in wait state.
00479
00480         Returns:
00481             tuple (error string, response string) from final control message
00482         """
00483         logging.info(self.logininfo + "disarm")
00484         self.ca.clearStatus()
00485         self.ca.armed = False
00486         control_messages = [
00487             ("HW_TRIG_EN", "0"),
00488             ("SW_TRIG_EN", "0"),
00489         ]
00490         return self.ca.submitMessages(control_messages, " disarm: ")
00491
```

6.7.3.6 dumpStatus()

```
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 824 of file [LLNL_v1.py](#).

```
00824     def dumpStatus(self):
00825         """
00826         Create dictionary of status values, DAC settings, monitor values, and register
00827         values
00828
00829         WARNING: the behavior of self-resetting subregisters may be difficult to predict
00830         and may generate contradictory results
00831
00832         Returns:
00833             dictionary of system diagnostic values
00834         """
00835         statusbits = self.checkStatus()
00836         statusbits2 = self.checkStatus2()
00837         temp = self.ca.getTemp()
00838
00839         statDict = OrderedDict(
00840             {
00841                 "Temperature reading": "{0:1.2f}".format(temp) + " C",
00842                 "Sensor read complete": str(statusbits[0]),
```

```

00843         "Coarse trigger detected": str(statusbits[1]),
00844         "Fine trigger detected": str(statusbits[2]),
00845         "Sensor readout in progress": str(statusbits[5]),
00846         "Sensor readout complete": str(statusbits[6]),
00847         "SRAM readout started": str(statusbits[7]),
00848         "SRAM readout complete": str(statusbits[8]),
00849         "High-speed timing configured": str(statusbits[9]),
00850         "All ADCs configured": str(statusbits[10]),
00851         "All pots configured": str(statusbits[11]),
00852         "HST_All_W_En detected": str(statusbits[12]),
00853         "Timer has reset": str(statusbits[13]),
00854         "Camera is Armed": str(statusbits[14]),
00855         "FPA_IF_TO": str(statusbits2[0]),
00856         "SRAM_RO_TO": str(statusbits2[1]),
00857         "PixelRd Timeout Error": str(statusbits2[2]),
00858         "UART_TX_TO_RST": str(statusbits2[3]),
00859         "UART_RX_TO_RST": str(statusbits2[4]),
00860     }
00861 )
00862
00863 POTDict = OrderedDict()
00864 MonDict = OrderedDict()
00865 for entry in self.subreg_aliases:
00866     if self.subreg_aliases[entry][0] == "P":
00867         val = str(round(self.ca.getPotV(entry), 3)) + " V"
00868         POTDict["POT_" + entry] = val
00869     else:
00870         val = str(round(self.ca.getMonV(entry), 3)) + " V"
00871         MonDict[entry] = val
00872
00873 regDict = OrderedDict()
00874 for key in self.registers.keys():
00875     err, rval = self.ca.getRegister(key)
00876     regDict[key] = rval
00877
00878 dumpDict = OrderedDict()
00879 for x in [statDict, MonDict, POTDict, regDict]:
00880     dumpDict.update(x)
00881 return dumpDict
00882
00883 """
00884 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00885 LLNL-CODE-838080
00886
00887 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00888 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00889 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00890 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00891 be made under this license.
00892 """

```

6.7.3.7 enableLED()

```

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 setSubregister())

Definition at line 603 of file LLNL_v1.py.

```

00603 def enableLED(self, status):
00604     """
00605     Enable/disable on-board LEDs

```

```

00606
00607     Args:
00608         status: 0 for disabled, 1 for enabled
00609
00610     Returns:
00611         tuple: (error string, response string from setSubregister())
00612     """
00613     logging.info(self.loginfo + "enableLED")
00614     if status:
00615         status = 1
00616     return self.ca.setSubregister("LED_EN", str(status))
00617

```

6.7.3.8 getPressure()

```

nsCamera.boards.LLNL_v1.llnl_v1.getPressure (
    self,
    offset,
    sensitivity,
    units )

```

Read pressure sensor

Currently unimplemented

Returns:
0 as float

Definition at line 700 of file [LLNL_v1.py](#).

```

00700     def getPressure(self, offset, sensitivity, units):
00701         """
00702         Read pressure sensor
00703
00704         Currently unimplemented
00705
00706         Returns:
00707             0 as float
00708         """
00709         logging.warning(
00710             "WARNING: [LLNL_v1] 'getPressure' is not implemented on the LLNLv1 board"
00711         )
00712         return 0.0
00713

```

6.7.3.9 getTemp()

```

nsCamera.boards.LLNL_v1.llnl_v1.getTemp (
    self,
    scale = None,
    offset = None,
    slope = None )

```

Read temperature sensor

Args:
scale: temperature scale to report (defaults to C, options are F and K)
offset: unused
slope: unused

Returns:
temperature as float on given scale

Definition at line 672 of file `LLNL_v1.py`.

```

00672     def getTemp(self, scale=None, offset=None, slope=None):
00673         """
00674         Read temperature sensor
00675         Args:
00676             scale: temperature scale to report (defaults to C, options are F and K)
00677             offset: unused
00678             slope: unused
00679         Returns:
00680             temperature as float on given scale
00681         """
00682         logging.debug(self.logdebug + "getTemp: scale = " + str(scale))
00683         err, rval = self.ca.getRegister("TEMP_SENSE_DATA")
00684         if err:
00685             logging.error(
00686                 self.logerr + "unable to retrieve temperature information ("
00687                     'getTemp), returning "0" '
00688             )
00689             return 0.0
00690
00691         ctemp = int(rval[-3:], 16) / 16.0
00692         if scale == "K":
00693             temp = ctemp + 273.15
00694         elif scale == "F":
00695             temp = 1.8 * ctemp + 32
00696         else:
00697             temp = ctemp
00698         return temp
00699

```

6.7.3.10 getTimer()

```

nsCamera.boards.LLNL_v1.llnl_v1.getTimer (
    self )

```

Read value of on-board timer

Returns:
timer value as integer

Definition at line 575 of file `LLNL_v1.py`.

```

00575     def getTimer(self):
00576         """
00577         Read value of on-board timer
00578
00579         Returns:
00580             timer value as integer
00581         """
00582         logging.info(self.loginfo + "getTimer")
00583         err, rval = self.ca.getRegister("TIMER_VALUE")
00584         if err:
00585             logging.error(
00586                 self.logerr + "unable to retrieve timer information (getTimer), "
00587                     'returning "0" '
00588             )
00589             return 0
00590         return int(rval, 16)
00591

```

6.7.3.11 initBoard()

```

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 315 of file [LLNL_v1.py](#).

```
00315     def initBoard(self):
00316         """
00317         Register and reset board, set up firmware for sensor
00318
00319         Returns:
00320             tuple (error string, response string) from final control message
00321         """
00322         logging.info(self.loginf + "initBoard LLNLv1")
00323         control_messages = [("LED_EN", "1")]
00324
00325         self.clearStatus()
00326         self.configADCs()
00327
00328         err, resp = self.ca.getSubregister("ADC5_VREF3")
00329         if err:
00330             logging.error(self.logerr + "unable to read 'ADC5_VREF3'")
00331         if int(resp, 2): # check to see if Vref is 3 or 2.5 volts
00332             vrefmax = 3.0
00333         else:
00334             vrefmax = 2.5
00335         err, resp = self.ca.getSubregister("ADC5_VREF")
00336         if err:
00337             logging.error(self.logerr + "unable to read 'ADC5_VREF'")
00338         self.VREF = vrefmax * int(resp, 2) / 1024.0
00339         err, multmask = self.ca.getSubregister("ADC5_MULT")
00340         if err:
00341             logging.error(self.logerr + "unable to read 'ADC5_MULT'")
00342         if multmask[0] and multmask[1] and multmask[3] and multmask[5]:
00343             self.ADC5_mult = 2
00344         elif not (multmask[0] or multmask[1] or multmask[3] or multmask[5]):
00345             self.ADC5_mult = 4
00346         else:
00347             logging.error(self.logerr + "inconsistent mode settings on ADC5")
00348         return self.ca.submitMessages(control_messages, " initBoard: ")
00349
```

6.7.3.12 initPots()

```
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 350 of file [LLNL_v1.py](#).

```
00350     def initPots(self):
00351         """
00352         Configure default pot settings before image acquisition
00353
00354         Returns:
00355             tuple (error string, response string) from final control message
00356         """
00357         logging.info(self.loginf + "initPots")
00358         if self.ca.sensorname == "icarus" or self.ca.sensorname == "icarus2":
00359             err0, _ = self.ca.setPot("HST_A_PDELAY", 0, errflag=True)
00360             err1, _ = self.ca.setPotV("HST_B_NDELAY", 3.3, errflag=True)
00361             err2, _ = self.ca.setPotV("HST_RO_IBIAS", 2.5, tune=True, errflag=True)
00362             err3, _ = self.ca.setPotV("HST_OSC_VREF_IN", 2.9, tune=True, errflag=True)
00363             err4, _ = self.ca.setPot("HST_B_PDELAY", 0, errflag=True)
00364             err5, _ = self.ca.setPotV("HST_OSC_CTL", 1.45, tune=True, errflag=True)
00365             err6, _ = self.ca.setPotV("HST_A_NDELAY", 3.3, errflag=True)
```

```

00366         err7, _ = self.ca.setPotV("VAB", 0.5, errflag=True)
00367         err8, _ = self.ca.setPotV("HST_RO_NC_IBIAS", 2.5, errflag=True)
00368         err9, _ = self.ca.setPotV("VRST", 0.3, tune=True, errflag=True)
00369         err = err0 + err1 + err2 + err3 + err4 + err5 + err6 + err7 + err8 + err9
00370     else: # Daedalus
00371         err0, _ = self.ca.setPotV("HST_OSC_CTL", 1.0, tune=True, errflag=True)
00372         err1, _ = self.ca.setPotV("HST_RO_NC_IBIAS", 1.0, errflag=True)
00373         err2, _ = self.ca.setPotV("HST_OSC_VREF_IN", 1.0, tune=True, errflag=True)
00374         err3, _ = self.ca.setPotV("VAB", 0.5, errflag=True)
00375         err = err0 + err1 + err2 + err3
00376     return err, ""
00377

```

6.7.3.13 initSensor()

```

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 404 of file [LLNL_v1.py](#).

```

00404     def initSensor(self):
00405         """
00406         Register sensor, set default timing settings
00407
00408         Returns:
00409             tuple (error string, response string) from final control message
00410         """
00411         logging.info(self.loginfo + "initSensor")
00412         if int(self.ca.FPGANum[7]) != self.ca.sensor.fpganumID:
00413             logging.error(
00414                 self.logerr + "unable to confirm sensor compatibility with FPGA"
00415             )
00416         self.registers.update(self.ca.sensor.sens_registers)
00417         self.subregisters.extend(self.ca.sensor.sens_subregisters)
00418         for s in self.ca.sensor.sens_subregisters:
00419             sr = SubRegister(
00420                 self,
00421                 name=s[0].upper(),
00422                 register=s[1].upper(),
00423                 start_bit=s[2],
00424                 width=s[3],
00425                 writable=s[4],
00426             )
00427             setattr(self, s[0].upper(), sr)
00428             self.subreglist.append(s[0])
00429         self.ca.checkSensorVoltStat()
00430         control_messages = self.ca.sensorSpecific() + [
00431             # ring w/caps=01, relax=00, ring w/o caps = 02
00432             ("OSC_SELECT", "00"),
00433             ("FPA_DIVCLK_EN_ADDR", "00000001"), # TODO Make this a subregister
00434         ]
00435         return self.ca.submitMessages(control_messages, " initSensor: ")
00436

```

6.7.3.14 latchPots()

```

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 378 of file [LLNL_v1.py](#).

```
00378     def latchPots(self):
00379         """
00380             Latch pot settings into sensor
00381
00382             Returns:
00383                 tuple (error string, response string) from final control message
00384             """
00385             logging.info(self.loginfo + "latchPots")
00386
00387             control_messages = [
00388                 ("POT_CTL", "00000003"), # latches register settings for pot 1
00389                 ("POT_CTL", "00000005"),
00390                 ("POT_CTL", "00000007"),
00391                 ("POT_CTL", "00000009"),
00392                 ("POT_CTL", "0000000B"),
00393                 ("POT_CTL", "0000000D"),
00394                 ("POT_CTL", "0000000F"),
00395                 ("POT_CTL", "00000011"),
00396                 ("POT_CTL", "00000013"),
00397                 ("POT_CTL", "00000015"),
00398                 ("POT_CTL", "00000017"),
00399                 ("POT_CTL", "00000019"),
00400                 ("POT_CTL", "0000001B"),
00401             ]
00402             return self.ca.submitMessages(control_messages, " latchPots: ")
00403
```

6.7.3.15 readSRAM()

```
nsCamera.boards.LLNL_v1.l1nl_v1.readSRAM (
    self )
```

Start readoff of SRAM

Returns:

tuple (error string, response string from register set)

Definition at line 529 of file [LLNL_v1.py](#).

```
00529     def readSRAM(self):
00530         """
00531             Start readoff of SRAM
00532
00533             Returns:
00534                 tuple (error string, response string from register set)
00535             """
00536             logging.info(self.loginfo + "readSRAM")
00537             control_messages = [("READ_SRAM", "1")]
00538             return self.ca.submitMessages(control_messages, " readSRAM: ")
00539
```

6.7.3.16 reportEdgeDetects()

```
nsCamera.boards.LLNL_v1.l1nl_v1.reportEdgeDetects (
    self )
```

Unimplemented

Definition at line 815 of file `LLNL_v1.py`.

```
00815     def reportEdgeDetects(self):
00816         """
00817         Unimplemented
00818         """
00819         logging.warning(
00820             self.logwarn + "'reportEdgeDetects' is not implemented on the LLNLv1 "
00821             "board "
00822         )
00823
```

6.7.3.17 reportStatus()

```
nsCamera.boards.LLNL_v1.llnl_v1.reportStatus (
    self )
```

Check contents of status register, print relevant messages

Definition at line 766 of file `LLNL_v1.py`.

```
00766     def reportStatus(self):
00767         """
00768         Check contents of status register, print relevant messages
00769         """
00770         statusbits = self.checkStatus()
00771         statusbits2 = self.checkStatus2()
00772         logging.info(self.loginfo + "Status report:")
00773         if int(statusbits[0]):
00774             print(self.loginfo + "Sensor read complete")
00775         if int(statusbits[1]):
00776             print(self.loginfo + "Coarse trigger detected")
00777         if int(statusbits[2]):
00778             print(self.loginfo + "Fine trigger detected")
00779         if int(statusbits[5]):
00780             print(self.loginfo + "Sensor readout in progress")
00781         if int(statusbits[6]):
00782             print(self.loginfo + "Sensor readout complete")
00783         if int(statusbits[7]):
00784             print(self.loginfo + "SRAM readout started")
00785         if int(statusbits[8]):
00786             print(self.loginfo + "SRAM readout complete")
00787         if int(statusbits[9]):
00788             print(self.loginfo + "High-speed timing configuration started")
00789         if int(statusbits[10]):
00790             print(self.loginfo + "All ADCs configured")
00791         if int(statusbits[11]):
00792             print(self.loginfo + "All pots configured")
00793         if int(statusbits[13]):
00794             print(self.loginfo + "Timer has reset")
00795         if int(statusbits[14]):
00796             print(self.loginfo + "Camera is Armed")
00797         self.ca.sensor.reportStatusSensor(statusbits, statusbits2)
00798         temp = int(statusbits[27:15:-1], 2) / 16.0
00799         logging.info(
00800             self.loginfo + "Temperature reading: " + "{0:1.2f}".format(temp) + " C"
00801         )
00802         # press = int(statusbits[:27:-1], 2)
00803         # logging.info(self.loginfo + "Pressure reading: " + "{0:1.2f}".format(press))
00804         if int(statusbits2[0]):
00805             print(self.loginfo + "FPA_IF_TO")
00806         if int(statusbits2[1]):
00807             print(self.loginfo + "SRAM_RO_TO")
00808         if int(statusbits2[2]):
00809             print(self.loginfo + "PixelRd Timeout Error")
00810         if int(statusbits2[3]):
00811             print(self.loginfo + "UART_TX_TO_RST")
00812         if int(statusbits2[4]):
00813             print(self.loginfo + "UART_RX_TO_RST")
00814
```

6.7.3.18 resetTimer()

```
nsCamera.boards.LLNL_v1.llnl_v1.resetTimer (
    self )
```

Reset on-board timer

Returns:
tuple (error string, response string from register set)

Definition at line 592 of file [LLNL_v1.py](#).

```
00592     def resetTimer(self):
00593         """
00594         Reset on-board timer
00595
00596         Returns:
00597             tuple (error string, response string from register set)
00598         """
00599         logging.info(self.logininfo + "resetTimer")
00600         control_messages = [("RESET_TIMER", "1"), ("RESET_TIMER", "0")]
00601         return self.ca.submitMessages(control_messages, " resetTimer: ")
00602
```

6.7.3.19 setLED()

```
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 setSubregister())

Definition at line 618 of file [LLNL_v1.py](#).

```
00618     def setLED(self, LED, status):
00619         """
00620         Illuminate on-board LED
00621
00622         Args:
00623             LED: LED number (1-8)
00624             status: 0 is off, 1 is on
00625
00626         Returns:
00627             tuple: (error string, response string from setSubregister())
00628         """
00629         logging.info(self.logininfo + "setLED")
00630         key = "LED" + str(LED)
00631         return self.ca.setSubregister(key, str(status))
00632
```

6.7.3.20 setPowerSave()

```
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 setSubregister())

Definition at line 633 of file LLNL_v1.py.

```
00633     def setPowerSave(self, status):
00634         """
00635         Select powersave option
00636
00637         Args:
00638             status: setting for powersave option (1 is enabled)
00639
00640         Returns:
00641             tuple (error string, response string from setSubregister())
00642         """
00643         logging.info(self.loginf + "setPowerSave")
00644         if status:
00645             status = 1
00646         return self.ca.setSubregister("POWERSAVE", str(status))
00647
```

6.7.3.21 setPPER()

```
nsCamera.boards.LLNL_v1.llnl_v1.setPPER (
    self,
    pollperiod )
```

Set polling period for ADCs.

Args:
pollperiod: milliseconds, between 1 and 255, defaults to 50

Returns:
tuple (error string, response string from setSubregister OR invalid time setting string)

Definition at line 648 of file LLNL_v1.py.

```
00648     def setPPER(self, pollperiod):
00649         """
00650         Set polling period for ADCs.
00651
00652         Args:
00653             pollperiod: milliseconds, between 1 and 255, defaults to 50
00654
00655         Returns:
00656             tuple (error string, response string from setSubregister OR invalid time
00657                 setting string)
00658         """
00659         logging.debug(self.logdebug + "setPPER: time = " + str(pollperiod))
00660         if pollperiod is None:
00661             pollperiod = 50
00662         if not isinstance(pollperiod, int) or pollperiod < 1 or pollperiod > 255:
00663             err = (
00664                 self.logerr + "invalid poll period submitted. Setting remains "
00665                 "unchanged. "
00666             )
00667             logging.error(err)
00668             return err, str(pollperiod)
00669         else:
00670             binset = bin(pollperiod)[2:].zfill(8)
00671             return self.ca.setSubregister("PPER", binset)
```

6.7.3.22 softReboot()

```
nsCamera.boards.LLNL_v1.llnl_v1.softReboot (
    self )
```

Perform software reboot of board. WARNING: board reboot will likely prevent correct communication responses and therefore will generate an error message

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

Definition at line 463 of file [LLNL_v1.py](#).

```
00463     def softReboot(self):
00464         """
00465         Perform software reboot of board. WARNING: board reboot will likely prevent
00466         correct communication responses and therefore will generate an error message
00467
00468         Returns:
00469         tuple (error string, response string) from final control message
00470         """
00471         logging.info(self.logininfo + "reboot")
00472         control_messages = [("RESET", "1")]
00473         return self.ca.submitMessages(control_messages, " disarm: ")
00474
```

6.7.3.23 startCapture()

```
nsCamera.boards.LLNL_v1.llnl_v1.startCapture (
    self,
    mode = "Hardware" )
```

Selects trigger mode and enables board for image capture

Args:
mode: trigger mode ("hardware"|"software"|"dual"|"h"|"s"|"d" , is case-insensitive)

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

Definition at line 492 of file [LLNL_v1.py](#).

```
00492     def startCapture(self, mode="Hardware"):
00493         """
00494         Selects trigger mode and enables board for image capture
00495
00496         Args:
00497         mode: trigger mode ("hardware"|"software"|"dual"|"h"|"s"|"d" , is case-
00498         insensitive)
00499
00500         Returns:
00501         tuple (error string, response string) from final control message
00502         """
00503         logging.info(self.logininfo + "startCapture")
00504         if self.ca.sensmanual:
00505             timingReg = "MANSHUT_MODE"
00506         else:
00507             timingReg = "HST_MODE"
00508
00509         if mode.upper()[0] == "S": # SOFTWARE
00510             trigmess = [
00511                 ("HW_TRIG_EN", "0"),
00512                 ("SW_TRIG_EN", "1"),
00513                 ("SW_TRIG_START", "1"),
```

```

00514         ]
00515     else: # HARDWARE
00516         trigmess = [
00517             ("SW_TRIG_EN", "0"),
00518             ("HW_TRIG_EN", "1"),
00519         ]
00520
00521     control_messages = [
00522         ("ADC_CTL", "0000001F"), # configure all ADCs
00523         (timingReg, "1"),
00524     ]
00525
00526     control_messages.extend(trigmess)
00527     return self.ca.submitMessages(control_messages, " startCapture: ")
00528

```

6.7.3.24 waitForSRAM()

```

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 540 of file [LLNL_v1.py](#).

```

00540     def waitForSRAM(self, timeout):
00541         """
00542         Wait until subreg 'SRAM_READY' flag is true or timeout is exceeded;
00543         timeout = None or zero means wait indefinitely
00544
00545         Args:
00546             timeout - time in seconds before readoff proceeds automatically without
00547             waiting for SRAM_READY flag
00548
00549         Returns:
00550             error string
00551         """
00552         logging.info(self.loginfo + "waitForSRAM")
00553         waiting = True
00554         starttime = time.time()
00555         err = ""
00556         while waiting:
00557             err, status = self.ca.getSubregister("SRAM_READY")
00558             if err:
00559                 logging.error(
00560                     self.logerr + "error in register read: " + err + " (waitForSRAM)"
00561                 )
00562             if int(status):
00563                 waiting = False
00564                 logging.info(self.loginfo + "SRAM ready")
00565             if self.ca.abort:
00566                 waiting = False
00567                 logging.info(self.loginfo + "readoff aborted by user")
00568                 self.ca.abort = False
00569             if timeout and time.time() - starttime > timeout:
00570                 err += self.logerr + "SRAM timeout; proceeding with download attempt"
00571                 logging.error(err)
00572                 return err
00573         return err
00574

```


6.7.4 Member Data Documentation

6.7.4.1 ADC5_bipolar

`nsCamera.boards.LLNL_v1.llnl_v1.ADC5_bipolar`

Definition at line 203 of file [LLNL_v1.py](#).

6.7.4.2 ADC5_mult

`nsCamera.boards.LLNL_v1.llnl_v1.ADC5_mult`

Definition at line 201 of file [LLNL_v1.py](#).

6.7.4.3 ca

`nsCamera.boards.LLNL_v1.llnl_v1.ca`

Definition at line 193 of file [LLNL_v1.py](#).

6.7.4.4 daedalus_monitor_controls

`nsCamera.boards.LLNL_v1.llnl_v1.daedalus_monitor_controls`

Definition at line 276 of file [LLNL_v1.py](#).

6.7.4.5 daedalus_subreg_aliases

`nsCamera.boards.LLNL_v1.llnl_v1.daedalus_subreg_aliases`

Definition at line 260 of file [LLNL_v1.py](#).

6.7.4.6 icarus_monitor_controls

`nsCamera.boards.LLNL_v1.llnl_v1.icarus_monitor_controls`

Definition at line 245 of file [LLNL_v1.py](#).

6.7.4.7 icarus_subreg_aliases

`nsCamera.boards.LLNL_v1.llnl_v1.icarus_subreg_aliases`

Definition at line 220 of file [LLNL_v1.py](#).

6.7.4.8 logcrit

```
nsCamera.boards.LLNL_v1.llnl_v1.logcrit
```

Definition at line 194 of file [LLNL_v1.py](#).

6.7.4.9 logdebug

```
nsCamera.boards.LLNL_v1.llnl_v1.logdebug
```

Definition at line 198 of file [LLNL_v1.py](#).

6.7.4.10 logerr

```
nsCamera.boards.LLNL_v1.llnl_v1.logerr
```

Definition at line 195 of file [LLNL_v1.py](#).

6.7.4.11 loginfo

```
nsCamera.boards.LLNL_v1.llnl_v1.loginfo
```

Definition at line 197 of file [LLNL_v1.py](#).

6.7.4.12 logwarn

```
nsCamera.boards.LLNL_v1.llnl_v1.logwarn
```

Definition at line 196 of file [LLNL_v1.py](#).

6.7.4.13 registers

```
nsCamera.boards.LLNL_v1.llnl_v1.registers [static]
```

Definition at line 37 of file [LLNL_v1.py](#).

6.7.4.14 rs422_baud

```
nsCamera.boards.LLNL_v1.llnl_v1.rs422_baud
```

Definition at line 204 of file [LLNL_v1.py](#).

6.7.4.15 rs422_cmd_wait

`nsCamera.boards.LLNL_v1.llnl_v1.rs422_cmd_wait`

Definition at line 205 of file [LLNL_v1.py](#).

6.7.4.16 subregisters

`list nsCamera.boards.LLNL_v1.llnl_v1.subregisters [static]`

Definition at line 98 of file [LLNL_v1.py](#).

6.7.4.17 subreglist

`nsCamera.boards.LLNL_v1.llnl_v1.subreglist`

Definition at line 287 of file [LLNL_v1.py](#).

6.7.4.18 VREF

`nsCamera.boards.LLNL_v1.llnl_v1.VREF`

Definition at line 200 of file [LLNL_v1.py](#).

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

- [C:/Users/hill35/git/camera_python/nsCamera/boards/LLNL_v1.py](#)

6.8 nsCamera.boards.LLNL_v4.IInl_v4 Class Reference

Public Member Functions

- [__init__](#) (self, camassem)
- [initBoard](#) (self)
- [initPots](#) (self)
- [latchPots](#) (self)
- [initSensor](#) (self)
- [configADCs](#) (self)
- [softReboot](#) (self)
- [disarm](#) (self)
- [startCapture](#) (self, mode="Hardware")
- [readSRAM](#) (self)
- [waitForSRAM](#) (self, timeout)
- [getTimer](#) (self)

- [resetTimer](#) (self)
- [enableLED](#) (self, status)
- [setLED](#) (self, LED, status)
- [setPowerSave](#) (self, status)
- [setPPER](#) (self, pollperiod)
- [getTemp](#) (self, scale=None)
- [getPressure](#) (self, offset=None, sensitivity=None, units=None)
- [clearStatus](#) (self)
- [checkStatus](#) (self)
- [checkStatus2](#) (self)
- [reportStatus](#) (self)
- [reportEdgeDetects](#) (self)
- [dumpStatus](#) (self)

Public Attributes

- [ca](#)
- [logcrit](#)
- [logerr](#)
- [logwarn](#)
- [loginfo](#)
- [logdebug](#)
- [VREF](#)
- [ADC5_mult](#)
- [ADC5_bipolar](#)
- [rs422_baud](#)
- [rs422_cmd_wait](#)
- [defoff](#)
- [defsens](#)
- [icarus_subreg_aliases](#)
- [icarus_monitor_controls](#)
- [daedalus_subreg_aliases](#)
- [daedalus_monitor_controls](#)
- [subreglist](#)

Static Public Attributes

- [registers](#)
- list [subregisters](#)

6.8.1 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](#).

6.8.2 Constructor & Destructor Documentation

6.8.2.1 __init__()

```
nsCamera.boards.LLNL_v4.llnl_v4.__init__ (
    self,
    camassem )
```

Definition at line 181 of file LLNL_v4.py.

```
00181     def __init__(self, camassem):
00182         self.ca = camassem
00183         self.logcrit = self.ca.logcritbase + "[LLNL_v4] "
00184         self.logerr = self.ca.logerrbase + "[LLNL_v4] "
00185         self.logwarn = self.ca.logwarnbase + "[LLNL_v4] "
00186         self.loginfo = self.ca.loginfobase + "[LLNL_v4] "
00187         self.logdebug = self.ca.logdebugbase + "[LLNL_v4] "
00188         logging.info(self.loginfo + "Initializing board object")
00189         self.VREF = 3.3 # must be supplied externally for ADC128S102
00190         self.ADC5_mult = 1
00191
00192         # ADC128S102; False => monitor range runs 0 to monmax, True => +/- monmax
00193         self.ADC5_bipolar = False
00194         self.rs422_baud = 921600
00195         self.rs422_cmd_wait = 0.3
00196
00197         fpgaNum_pkt = Packet(cmd="1", addr=self.registers["FPGA_NUM"])
00198         fpgaRev_pkt = Packet(cmd="1", addr=self.registers["FPGA_REV"])
00199
00200         _, _ = self.ca.sendCMD(fpgaNum_pkt) # dummy duplicate call
00201         err, rval = self.ca.sendCMD(fpgaNum_pkt)
00202         self.ca.FPGA_NUM = rval[8:16]
00203
00204         err, rval = self.ca.sendCMD(fpgaRev_pkt)
00205         self.ca.FPGA_VERSION = rval[8:16]
00206
00207         self.defoff = 34.5 # default pressure sensor offset
00208         self.defsens = 92.5 # default pressure sensor sensitivity
00209
00210         # TODO: move to sensor scripts?
00211         # map channels to signal names for abstraction at the camera assembler level;
00212         # each requires a corresponding entry in 'subregisters'
00213         self.icarus_subreg_aliases = OrderedDict(
00214             {
00215                 "HST_A_PDELAY": "DACA",
00216                 "HST_A_NDELAY": "DACB",
00217                 "HST_B_PDELAY": "DACC",
00218                 "HST_B_NDELAY": "DACD",
00219                 "HST_RO_IBIAS": "DACE",
00220                 "HST_RO_NC_IBIAS": "DACE",
00221                 "HST_OSC_CTL": "DACF",
00222                 "VAB": "DACG",
00223                 "VRST": "DACH",
00224                 "MON_PRES_MINUS": "MON_CH1",
00225                 "MON_PRES_PLUS": "MON_CH2",
00226                 "MON_TEMP": "MON_CH3",
00227                 "MON_COL_TOP_IBIAS_IN": "MON_CH4",
00228                 "MON_HST_OSC_R_BIAS": "MON_CH5",
00229                 "MON_VAB": "MON_CH6",
00230                 "MON_HST_RO_IBIAS": "MON_CH7",
00231                 "MON_HST_RO_NC_IBIAS": "MON_CH7",
00232                 "MON_VRST": "MON_CH8",
00233                 "MON_COL_BOT_IBIAS_IN": "MON_CH9",
00234                 "MON_HST_A_PDELAY": "MON_CH10",
00235                 "MON_HST_B_NDELAY": "MON_CH11",
00236                 "DOSIMETER": "MON_CH12",
00237                 "MON_HST_OSC_VREF_IN": "MON_CH13",
00238                 "MON_HST_B_PDELAY": "MON_CH14",
00239                 "MON_HST_OSC_CTL": "MON_CH15",
00240                 "MON_HST_A_NDELAY": "MON_CH16",
00241                 "MON_CHA": "MON_CH10",
00242                 "MON_CHB": "MON_CH16",
00243                 "MON_CHC": "MON_CH14",
00244                 "MON_CHD": "MON_CH11",
00245                 "MON_CHE": "MON_CH7",
00246                 "MON_CHE": "MON_CH15",
```

```

00247         "MON_CHG": "MON_CH6",
00248         "MON_CHH": "MON_CH8",
00249     }
00250 )
00251 # Read-only; identifies controls corresponding to monitors
00252 self.icarus_monitor_controls = OrderedDict(
00253     {
00254         "MON_CH10": "DACA",
00255         "MON_CH16": "DACB",
00256         "MON_CH14": "DACC",
00257         "MON_CH11": "DADC",
00258         "MON_CH7": "DACE",
00259         "MON_CH15": "DACF",
00260         "MON_CH6": "DACG",
00261         "MON_CH8": "DACH",
00262     }
00263 )
00264 self.daedalus_subreg_aliases = OrderedDict(
00265     {
00266         "HST_OSC_VREF_IN": "DACC",
00267         "HST_OSC_CTL": "DACE",
00268         "COL_TST_IN": "DACF",
00269         "VAB": "DACG",
00270         "VRST": "DACH",
00271         "MON_PRES_MINUS": "MON_CH1",
00272         "MON_PRES_PLUS": "MON_CH2",
00273         "MON_TEMP": "MON_CH3",
00274         "MON_VAB": "MON_CH6",
00275         "MON_HST_OSC_CTL": "MON_CH7",
00276         "MON_TSENSE_OUT": "MON_CH10",
00277         "MON_BGREF": "MON_CH11",
00278         "DOSIMETER": "MON_CH12",
00279         "MON_HST_RO_NC_IBIAS": "MON_CH13",
00280         "MON_HST_OSC_VREF_IN": "MON_CH14",
00281         "MON_COL_TST_IN": "MON_CH15",
00282         "MON_HST_OSC_PBIAS_PAD": "MON_CH16",
00283         "MON_CHC": "MON_CH14",
00284         "MON_CHE": "MON_CH7",
00285         "MON_CHF": "MON_CH15",
00286         "MON_CHG": "MON_CH6",
00287         "MON_CHH": "MON_CH8",
00288     }
00289 )
00290 # Read-only; identifies controls corresponding to monitors
00291 self.daedalus_monitor_controls = OrderedDict(
00292     {
00293         "MON_CH14": "DACC",
00294         "MON_CH7": "DACE",
00295         "MON_CH15": "DACF",
00296         "MON_CH6": "DACG",
00297         "MON_CH8": "DACH",
00298     }
00299 )
00300 self.subreglist = []
00301 for s in self.subregisters:
00302     self.subreglist.append(s[0].upper())
00303     sr = SubRegister(
00304         self,
00305         name=s[0].upper(),
00306         register=s[1].upper(),
00307         start_bit=s[2],
00308         width=s[3],
00309         writable=s[4],
00310     )
00311     setattr(self, s[0].upper(), sr)
00312
00313 # set voltage ranges for all DACs - WARNING: actual output voltage limited to
00314 # external supply (3.3 V)
00315 # setpot('potx', n) will generate 3.3 V for all n > .66
00316 for n in range(0, 8):
00317     potname = "DAC" + string.ascii_uppercase[n]
00318     potobj = getattr(self, potname)
00319     potobj.minV = 0
00320     potobj.maxV = 5 #
00321     potobj.resolution = (
00322         1.0 * potobj.maxV - potobj.minV
00323     ) / potobj.max_value # 76 uV / LSB
00324

```

6.8.3 Member Function Documentation

6.8.3.1 checkStatus()

```
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 706 of file LLNL_v4.py.

```
00706     def checkStatus(self):
00707         """
00708         Check status register, convert to reverse-order bit stream (i.e., bit 0 is
00709         statusbits[0])
00710
00711         Returns:
00712             bit string (no '0b') in reversed order
00713         """
00714         err, rval = self.ca.getRegister("STAT_REG")
00715         rvalbits = bin(int(rval, 16))[2:].zfill(32)
00716         statusbits = rvalbits[::-1]
00717         return statusbits
00718
```

6.8.3.2 checkStatus2()

```
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 719 of file LLNL_v4.py.

```
00719     def checkStatus2(self):
00720         """
00721         Check second status register, convert to reverse-order bit stream (i.e., bit 0
00722         is statusbits[0])
00723
00724         Returns: bit string (no '0b') in reversed order
00725         """
00726         err, rval = self.ca.getRegister("STAT_REG2")
00727         rvalbits = bin(int(rval, 16))[2:].zfill(6)
00728         statusbits = rvalbits[::-1]
00729         return statusbits
00730
```

6.8.3.3 clearStatus()

```
nsCamera.boards.LLNL_v4.llnl_v4.clearStatus (
    self )
```

Check status registers to clear them

Returns:
error string

Definition at line 692 of file [LLNL_v4.py](#).

```
00692     def clearStatus(self):
00693         """
00694         Check status registers to clear them
00695
00696         Returns:
00697             error string
00698         """
00699         err1, rval = self.ca.getRegister("STAT_REG_SRC")
00700         err2, rval = self.ca.getRegister("STAT_REG2_SRC")
00701         err = err1 + err2
00702         if err:
00703             logging.error(self.logerr + "clearStatus failed")
00704         return err
00705
```

6.8.3.4 configADCs()

```
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 402 of file [LLNL_v4.py](#).

```
00402     def configADCs(self):
00403         """
00404         Sets default ADC configuration (does not latch settings)
00405
00406         Returns:
00407             tuple (error string, response string) from final control message
00408         """
00409         logging.info(self.loginf + "configADCs")
00410
00411         control_messages = [
00412             # just in case ADC_RESET was set on any of the ADCs (pull all ADCs out of
00413             # reset)
00414             ("ADC_RESET", "00000000"),
00415             # workaround for uncertain behavior after previous readoff
00416             ("ADC1_CONFIG_DATA", "FFFFFFFF"),
00417             ("ADC2_CONFIG_DATA", "FFFFFFFF"),
00418             ("ADC3_CONFIG_DATA", "FFFFFFFF"),
00419             ("ADC4_CONFIG_DATA", "FFFFFFFF"),
00420             ("ADC_CTL", "FFFFFFFF"),
00421             ("ADC1_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
00422             ("ADC2_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
00423             ("ADC3_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
00424             ("ADC4_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
00425         ]
00426         return self.ca.submitMessages(control_messages, " configADCs: ")
00427
```


6.8.3.5 disarm()

```
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 440 of file [LLNL_v4.py](#).

```
00440     def disarm(self):
00441         """
00442             Takes camera out of trigger wait state. Has no effect if camera is not in wait
00443             state.
00444
00445             Returns:
00446                 tuple (error string, response string) from final control message
00447         """
00448         logging.info(self.logininfo + "disarm")
00449         self.ca.clearStatus()
00450         self.ca.armed = False
00451         control_messages = [
00452             ("HW_TRIG_EN", "0"),
00453             ("SW_TRIG_EN", "0"),
00454         ]
00455         self.ca.comms.skipError = False
00456         return self.ca.submitMessages(control_messages, " disarm: ")
00457
```

6.8.3.6 dumpStatus()

```
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 821 of file [LLNL_v4.py](#).

```
00821     def dumpStatus(self):
00822         """
00823             Create dictionary of status values, DAC settings, monitor values, and register
00824             values.
00825
00826             Returns:
00827                 dictionary of system diagnostic values
00828         """
00829         statusbits = self.checkStatus()
00830         statusbits2 = self.checkStatus2()
00831
00832         temp = int(statusbits[23:16:-1], 2) * 3.3 * 1000 / 4096
00833         press = int(statusbits[:23:-1], 2) * 3.3 * 1000 / 4096
00834
00835         statDict = OrderedDict(
00836             {
00837                 "Temperature sensor reading (Deg C)": "{0:1.2f}".format(temp),
00838                 "Pressure reading (Torr)": str(round(self.ca.getPressure(), 3)),
00839                 "Pressure sensor reading (mV)": "{0:1.2f}".format(press),
00840                 "Sensor read complete": str(statusbits[0]),
00841                 "Coarse trigger detected": str(statusbits[1]),
```

```

00842         "Fine trigger detected": str(statusbits[2]),
00843         "Sensor readout in progress": str(statusbits[5]),
00844         "Sensor readout complete": str(statusbits[6]),
00845         "SRAM readout started": str(statusbits[7]),
00846         "SRAM readout complete": str(statusbits[8]),
00847         "High-speed timing configured": str(statusbits[9]),
00848         "All ADCs configured": str(statusbits[10]),
00849         "All DACs configured": str(statusbits[11]),
00850         "Timer has reset": str(statusbits[13]),
00851         "Camera is Armed": str(statusbits[14]),
00852         "FPA_IF_TO": str(statusbits2[0]),
00853         "SRAM_RO_TO": str(statusbits2[1]),
00854         "PixelRd Timeout Error": str(statusbits2[2]),
00855         "UART_TX_TO_RST": str(statusbits2[3]),
00856         "UART_RX_TO_RST": str(statusbits2[4]),
00857         "PDBIAS Unready": str(statusbits2[5]),
00858     }
00859 )
00860
00861     if self.ca.sensorname == "icarus" or self.ca.sensorname == "icarus2":
00862         senslabs = {
00863             3: "W3_Top_A_Edge1 detected",
00864             4: "W3_Top_B_Edge1 detected",
00865             12: "HST_All_W_En detected",
00866         }
00867     else:
00868         senslabs = {
00869             3: "RSLROWOUTA",
00870             4: "RSLROWOUTB",
00871             12: "RSLNALLWENB",
00872             15: "RSLNALLWENA",
00873             16: "Config HST is done",
00874         }
00875     sensDict = {senslabs[x]: str(statusbits[x]) for x in senslabs.keys()}
00876
00877     DACDict = OrderedDict()
00878     MonDict = OrderedDict()
00879     for entry in self.subreg_aliases:
00880         if self.subreg_aliases[entry][0] == "D":
00881             val = str(round(self.ca.getPotV(entry), 3))
00882             DACDict["DAC_" + entry] = val
00883         else:
00884             val = str(round(self.ca.getMonV(entry), 3))
00885             MonDict[entry] = val
00886
00887     regDict = OrderedDict()
00888     for key in self.registers.keys():
00889         # Load in all registers except for the read-clear status registers.
00890         if key == "STAT_REG_SRC" or key == "STAT_REG2_SRC":
00891             pass
00892         else:
00893             err, rval = self.ca.getRegister(key)
00894             regDict[key] = rval
00895
00896     dumpDict = OrderedDict()
00897     for x in [statDict, sensDict, MonDict, DACDict, regDict]:
00898         dumpDict.update(x)
00899     return dumpDict
00900
00901
00902 """
00903 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00904 LLNL-CODE-838080
00905
00906 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00907 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00908 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00909 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00910 be made under this license.
00911 """

```

6.8.3.7 enableLED()

```

nsCamera.boards.LLNL_v4.llnl_v4.enableLED (
    self,
    status )

```

Dummy function; feature is not implemented on LLNL_V4 board

Returns:

tuple: dummy of (error string, response string from setSubregister())

Definition at line 571 of file [LLNL_v4.py](#).

```
00571     def enableLED(self, status):
00572         """
00573         Dummy function; feature is not implemented on LLNL_V4 board
00574
00575         Returns:
00576             tuple: dummy of (error string, response string from setSubregister())
00577         """
00578         del status
00579         return "", "0"
00580
```

6.8.3.8 getPressure()

```
nsCamera.boards.LLNL_v4.llnl_v4.getPressure (
    self,
    offset = None,
    sensitivity = None,
    units = None )
```

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 653 of file [LLNL_v4.py](#).

```
00653     def getPressure(self, offset=None, sensitivity=None, units=None):
00654         """
00655         Read pressure sensor. Uses default offset and sensitivity defined in init
00656         function unless alternatives are specified. NOTE: to reset defaults, reassign
00657         board.defoff and board.defsens explicitly
00658
00659         Args:
00660             offset: non-default offset in mv/V
00661             sensitivity: non-default sensitivity in mV/V/span
00662             units: units to report pressure (defaults to Torr, options are psi, bar,
00663                   inHg, atm)
00664
00665         Returns:
00666             Pressure as float in chosen units, defaults to torr
00667         """
00668         if offset is None:
00669             offset = self.defoff
00670         if sensitivity is None:
00671             sensitivity = self.defsens
00672         if units is None:
00673             units = "torr"
00674         pplus = self.ca.getMonV("MON_PRES_PLUS")
00675         pminus = self.ca.getMonV("MON_PRES_MINUS")
00676         delta = 1000 * (pplus - pminus)
00677         ratio = sensitivity / 30 # nominal is 21/30
00678         psi = (delta - offset) / ratio
00679         if units.lower() == "psi":
```

```

00680         press = psi
00681     elif units.lower() == "bar":
00682         press = psi / 14.504
00683     elif units.lower() == "atm":
00684         press = psi / 14.695
00685     elif units.lower() == "inHg":
00686         press = psi * 2.036
00687     else:
00688         press = 51.715 * psi # default to Torr
00689
00690     return press
00691

```

6.8.3.9 getTemp()

```

nsCamera.boards.LLNL_v4.llnl_v4.getTemp (
    self,
    scale = None )

```

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 628 of file [LLNL_v4.py](#).

```

00628     def getTemp(self, scale=None):
00629         """
00630         Read temperature sensor
00631         Args:
00632             scale: temperature scale to report (defaults to C, options are F and K)
00633
00634         Returns:
00635             temperature as float on given scale
00636         """
00637         err, rval = self.ca.getMonV("MON_TEMP", errflag=True)
00638         if err:
00639             logging.error(
00640                 self.logerr + "unable to retrieve temperature information ("
00641                 'getTemp), returning "0" '
00642             )
00643             return 0.0
00644         ctemp = rval * 1000 - 273.15
00645         if scale == "K":
00646             temp = ctemp + 273.15
00647         elif scale == "F":
00648             temp = 1.8 * ctemp + 32
00649         else:
00650             temp = ctemp
00651         return temp
00652

```

6.8.3.10 getTimer()

```

nsCamera.boards.LLNL_v4.llnl_v4.getTimer (
    self )

```

Read value of on-board timer

Returns:

timer value as integer

Definition at line 543 of file LLNL_v4.py.

```
00543     def getTimer(self):
00544         """
00545         Read value of on-board timer
00546
00547         Returns:
00548             timer value as integer
00549         """
00550         logging.info(self.loginf + "getTimer")
00551         err, rval = self.ca.getRegister("TIMER_VALUE")
00552         if err:
00553             logging.error(
00554                 self.logerr + "unable to retrieve timer information (getTimer), "
00555                 'returning "0" '
00556             )
00557         return 0
00558     return int(rval, 16)
00559
```

6.8.3.11 initBoard()

```
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 325 of file LLNL_v4.py.

```
00325     def initBoard(self):
00326         """
00327         Register and reset board, set up firmware for sensor
00328
00329         Returns:
00330             tuple (error string, response string) from final control message
00331         """
00332         logging.info(self.loginf + "initBoard LLNLv4")
00333         control_messages = []
00334         self.clearStatus()
00335         self.configADCs()
00336         return self.ca.submitMessages(control_messages, " initBoard: ")
00337
```

6.8.3.12 initPots()

```
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 338 of file LLNL_v4.py.

```
00338     def initPots(self):
00339         """
00340         Dummy function; initial DAC values are set by firmware at startup
00341
00342         Returns:
00343             tuple (empty string, empty string)
00344         """
00345         logging.debug(self.logdebug + "InitPots")
00346         return "", ""
00347
```

6.8.3.13 initSensor()

```
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 368 of file [LLNL_v4.py](#).

```
00368     def initSensor(self):
00369         """
00370         Register sensor, set default timing settings
00371
00372         Returns:
00373             tuple (error string, response string) from final control message
00374         """
00375         logging.info(self.loginfo + "initSensor")
00376         if int(self.ca.FPGANum[7]) != self.ca.sensor.fpganumID:
00377             logging.warning(
00378                 self.logwarn + "unable to confirm sensor compatibility with FPGA"
00379             )
00380         self.registers.update(self.ca.sensor.sens_registers)
00381         self.subregisters.extend(self.ca.sensor.sens_subregisters)
00382         for s in self.ca.sensor.sens_subregisters:
00383             sr = SubRegister(
00384                 self,
00385                 name=s[0].upper(),
00386                 register=s[1].upper(),
00387                 start_bit=s[2],
00388                 width=s[3],
00389                 writable=s[4],
00390             )
00391             setattr(self, s[0].upper(), sr)
00392             self.subreglist.append(s[0])
00393         # TODO: self.ca.checkSensorVoltStat() # SENSOR_VOLT_STAT and SENSOR_VOLT_CTL are
00394         # deactivated for v4 icarus and daedalus firmware for now, is this permanent?
00395         control_messages = self.ca.sensorSpecific() + [
00396             # ring w/caps=01, relax=00, ring w/o caps = 02
00397             ("OSC_SELECT", "00"),
00398             ("FPA_DIVCLK_EN_ADDR", "00000001"),
00399         ]
00400         return self.ca.submitMessages(control_messages, " initSensor: ")
00401
```

6.8.3.14 latchPots()

```
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 348 of file [LLNL_v4.py](#).

```
00348     def latchPots(self):
00349         """
00350         Latch DAC settings into sensor
00351
00352         Returns:
00353             tuple (error string, response string) from final control message
00354         """
00355         logging.info(self.loginfo + "latchPots")
```

```

00356         control_messages = [
00357             ("DAC_CTL", "00000001"), # latches register settings for DACA
00358             ("DAC_CTL", "00000003"),
00359             ("DAC_CTL", "00000005"),
00360             ("DAC_CTL", "00000007"),
00361             ("DAC_CTL", "00000009"),
00362             ("DAC_CTL", "0000000B"),
00363             ("DAC_CTL", "0000000D"),
00364             ("DAC_CTL", "0000000F"),
00365         ]
00366         return self.ca.submitMessages(control_messages, " latchPots: ")
00367

```

6.8.3.15 readSRAM()

```

nsCamera.boards.LLNL_v4.llnl_v4.readSRAM (
    self )

```

Start readoff of SRAM

Returns:

tuple (error string, response string from register set)

Definition at line 495 of file [LLNL_v4.py](#).

```

00495     def readSRAM(self):
00496         """
00497         Start readoff of SRAM
00498
00499         Returns:
00500             tuple (error string, response string from register set)
00501         """
00502         logging.info(self.logininfo + "readSRAM")
00503         control_messages = [("READ_SRAM", "1")]
00504         return self.ca.submitMessages(control_messages, " readSRAM: ")
00505

```

6.8.3.16 reportEdgeDetects()

```

nsCamera.boards.LLNL_v4.llnl_v4.reportEdgeDetects (
    self )

```

Report edge detects

Definition at line 788 of file [LLNL_v4.py](#).

```

00788     def reportEdgeDetects(self):
00789         """
00790         Report edge detects
00791         """
00792         err, rval = self.ca.getRegister("STAT_EDGE_DETECTS")
00793         # shift to left to fake missing edge detect
00794         edgebits = bin(int(rval, 16) << 1)[2:].zfill(32)
00795         # reverse to get order matching assignment
00796         bitsrev = edgebits[::-1]
00797         detdict = {}
00798         bitidx = 0
00799         for frame in range(4):
00800             for vert in ("TOP", "BOT"):
00801                 for edge in range(1, 3):
00802                     for hor in ("A", "B"):
00803                         detname = (
00804                             "W"
00805                             + str(frame)

```

```

00806             + "_"
00807             + vert
00808             + "_"
00809             + hor
00810             + "_EDGE"
00811             + str(edge)
00812         )
00813         detdict[detname] = bitsrev[bitidx]
00814         bitidx += 1
00815     # remove faked detect
00816     del detdict["W0_TOP_A_EDGE1"]
00817     logging.info(self.loginfo + "Edge detect report:")
00818     for key, val in detdict.items():
00819         logging.info(self.loginfo + key + ": " + val)
00820

```

6.8.3.17 reportStatus()

```

nsCamera.boards.LLNL_v4.llnl_v4.reportStatus (
    self )

```

Check contents of status register, print relevant messages

Definition at line 731 of file LLNL_v4.py.

```

00731     def reportStatus(self):
00732         """
00733         Check contents of status register, print relevant messages
00734         """
00735         statusbits = self.checkStatus()
00736         statusbits2 = self.checkStatus2()
00737
00738         logging.info(self.loginfo + "Status report:")
00739         if int(statusbits[0]):
00740             print(self.loginfo + "Sensor read complete")
00741         if int(statusbits[1]):
00742             print(self.loginfo + "Coarse trigger detected")
00743         if int(statusbits[2]):
00744             print(self.loginfo + "Fine trigger detected")
00745         if int(statusbits[5]):
00746             print(self.loginfo + "Sensor readout in progress")
00747         if int(statusbits[6]):
00748             print(self.loginfo + "Sensor readout complete")
00749         if int(statusbits[7]):
00750             print(self.loginfo + "SRAM readout started")
00751         if int(statusbits[8]):
00752             print(self.loginfo + "SRAM readout complete")
00753         if int(statusbits[9]):
00754             print(self.loginfo + "High-speed timing configuration started")
00755         if int(statusbits[10]):
00756             print(self.loginfo + "All ADCs configured")
00757         if int(statusbits[11]):
00758             print(self.loginfo + "All DACs configured")
00759         if int(statusbits[13]):
00760             print(self.loginfo + "Timer has reset")
00761         if int(statusbits[14]):
00762             print(self.loginfo + "Camera is Armed")
00763         if int(statusbits[16]):
00764             print(self.loginfo + "High-speed timing configuration complete")
00765         self.ca.sensor.reportStatusSensor(statusbits, statusbits2)
00766         temp = int(statusbits[23:16:-1], 2) * 3.3 * 1000 / 4096
00767         logging.info(
00768             self.loginfo + "Temperature reading: " + "{0:1.2f}".format(temp) + " C"
00769         )
00770         press = int(statusbits[:23:-1], 2) * 3.3 * 1000 / 4096
00771         logging.info(
00772             self.loginfo
00773             + "Pressure sensor reading: "
00774             + "{0:1.2f}".format(press)
00775             + " mV"
00776         )
00777         if int(statusbits2[0]):
00778             print(self.loginfo + "FPA_IF_TO")

```



```

00779         if int(statusbits2[1]):
00780             print(self.loginfo + "SRAM_RO_TO")
00781         if int(statusbits2[2]):
00782             print(self.loginfo + "PixelRd Timeout Error")
00783         if int(statusbits2[3]):
00784             print(self.loginfo + "UART_TX_TO_RST")
00785         if int(statusbits2[4]):
00786             print(self.loginfo + "UART_RX_TO_RST")
00787

```

6.8.3.18 resetTimer()

```

nsCamera.boards.LLNL_v4.llnl_v4.resetTimer (
    self )

```

Reset on-board timer

Returns:
tuple (error string, response string from register set)

Definition at line 560 of file [LLNL_v4.py](#).

```

00560     def resetTimer(self):
00561         """
00562         Reset on-board timer
00563
00564         Returns:
00565             tuple (error string, response string from register set)
00566         """
00567         logging.info(self.loginfo + "resetTimer")
00568         control_messages = [("RESET_TIMER", "1"), ("RESET_TIMER", "0")]
00569         return self.ca.submitMessages(control_messages, " resetTimer: ")
00570

```

6.8.3.19 setLED()

```

nsCamera.boards.LLNL_v4.llnl_v4.setLED (
    self,
    LED,
    status )

```

Dummy function; feature is not implemented on LLNL_V4 board

Returns:
tuple: dummy of (error string, response string from setSubregister())

Definition at line 581 of file [LLNL_v4.py](#).

```

00581     def setLED(self, LED, status):
00582         """
00583         Dummy function; feature is not implemented on LLNL_V4 board
00584
00585         Returns:
00586             tuple: dummy of (error string, response string from setSubregister())
00587         """
00588         del LED, status
00589         return "", "0"
00590

```

6.8.3.20 setPowerSave()

```
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 setSubregister())

Definition at line 591 of file [LLNL_v4.py](#).

```
00591     def setPowerSave(self, status):
00592         """
00593         Select powersave option
00594
00595         Args:
00596             status: setting for powersave option (1 is enabled)
00597
00598         Returns:
00599             tuple (error string, response string from setSubregister())
00600         """
00601         if status:
00602             status = 1
00603         return self.ca.setSubregister("POWERSAVE", str(status))
00604
```

6.8.3.21 setPPER()

```
nsCamera.boards.LLNL_v4.llnl_v4.setPPER (
    self,
    pollperiod )
```

Set polling period for ADCs.

Args:
pollperiod: milliseconds, between 1 and 255; defaults to 50

Returns:
tuple (error string, response string from setSubregister() OR invalid time setting string)

Definition at line 605 of file [LLNL_v4.py](#).

```
00605     def setPPER(self, pollperiod):
00606         """
00607         Set polling period for ADCs.
00608         Args:
00609             pollperiod: milliseconds, between 1 and 255; defaults to 50
00610
00611         Returns:
00612             tuple (error string, response string from setSubregister() OR invalid time
00613             setting string)
00614         """
00615         if pollperiod is None:
00616             pollperiod = 50
00617         if not isinstance(pollperiod, int) or pollperiod < 1 or pollperiod > 255:
00618             err = (
00619                 self.logerr + "invalid poll period submitted. Setting remains "
00620                 "unchanged. "
00621             )
00622             logging.error(err)
00623             return err, str(pollperiod)
00624         else:
00625             binset = bin(pollperiod)[2:].zfill(8)
00626             return self.ca.setSubregister("PPER", binset)
00627
```

6.8.3.22 softReboot()

```
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 428 of file [LLNL_v4.py](#).

```
00428     def softReboot(self):
00429         """
00430         Perform software reboot of board. WARNING: board reboot will likely prevent
00431         correct response and therefore will generate an error message
00432
00433         Returns:
00434         tuple (error string, response string) from final control message
00435         """
00436         logging.info(self.logininfo + "reboot")
00437         control_messages = [("RESET", "0")]
00438         return self.ca.submitMessages(control_messages, " disarm: ")
00439
```

6.8.3.23 startCapture()

```
nsCamera.boards.LLNL_v4.llnl_v4.startCapture (
    self,
    mode = "Hardware" )
```

Selects trigger mode and enables board for image capture

Args:

mode: trigger mode ("hardware"|"software"|"dual"|"h"|"s"|"d" , is case-insensitive)

Returns:

tuple (error string, response string) from final control message

Definition at line 458 of file [LLNL_v4.py](#).

```
00458     def startCapture(self, mode="Hardware"):
00459         """
00460         Selects trigger mode and enables board for image capture
00461
00462         Args:
00463         mode: trigger mode ("hardware"|"software"|"dual"|"h"|"s"|"d" , is case-
00464         insensitive)
00465
00466         Returns:
00467         tuple (error string, response string) from final control message
00468         """
00469         logging.info(self.logininfo + "startCapture")
00470         if self.ca.sensmanual:
00471             timingReg = "MANSHUT_MODE"
00472         else:
00473             timingReg = "HST_MODE"
00474
00475         if mode.upper()[0] == "S": # SOFTWARE
00476             trigmess = [
00477                 ("HW_TRIG_EN", "0"),
00478                 ("SW_TRIG_EN", "1"),
00479                 ("SW_TRIG_START", "1"),
```

```

00480         ]
00481     else: # HARDWARE
00482         trigmess = [
00483             ("SW_TRIG_EN", "0"),
00484             ("HW_TRIG_EN", "1"),
00485         ]
00486
00487         control_messages = [
00488             ("ADC_CTL", "0000000F"), # configure all ADCs
00489             (timingReg, "1"),
00490         ]
00491
00492         control_messages.extend(trigmess)
00493     return self.ca.submitMessages(control_messages, " startCapture: ")
00494

```

6.8.3.24 waitForSRAM()

```

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 506 of file [LLNL_v4.py](#).

```

00506     def waitForSRAM(self, timeout):
00507         """
00508         Wait until subreg 'SRAM_READY' flag is true or timeout is exceeded;
00509         timeout = None or zero means wait indefinitely
00510
00511         Args:
00512             timeout - time in seconds before readoff proceeds automatically without
00513             waiting for SRAM_READY flag
00514
00515         Returns:
00516             error string
00517         """
00518         logging.info(self.loginfo + "waitForSRAM, timeout = " + str(timeout))
00519         waiting = True
00520         starttime = time.time()
00521         err = ""
00522         while waiting:
00523             err, status = self.ca.getSubregister("SRAM_READY")
00524             if err:
00525                 err = self.logerr + "error in register read: " + err + " (waitForSRAM)"
00526                 logging.error(err)
00527             if int(status):
00528                 waiting = False
00529                 logging.info(self.loginfo + "SRAM ready")
00530             if self.ca.abort:
00531                 waiting = False
00532                 logging.info(self.loginfo + "readoff aborted by user")
00533                 self.ca.abort = False
00534             if timeout and time.time() - starttime > timeout:
00535                 err += self.logerr + "SRAM timeout; proceeding with download attempt"
00536                 logging.error(err)
00537                 return err
00538             # Slow down for debugging (avoid thousands of messages)
00539             if self.ca.verbose >= 5:
00540                 time.sleep(0.5)
00541         return err
00542

```

6.8.4 Member Data Documentation

6.8.4.1 ADC5_bipolar

`nsCamera.boards.LLNL_v4.llnl_v4.ADC5_bipolar`

Definition at line 193 of file [LLNL_v4.py](#).

6.8.4.2 ADC5_mult

`nsCamera.boards.LLNL_v4.llnl_v4.ADC5_mult`

Definition at line 190 of file [LLNL_v4.py](#).

6.8.4.3 ca

`nsCamera.boards.LLNL_v4.llnl_v4.ca`

Definition at line 182 of file [LLNL_v4.py](#).

6.8.4.4 daedalus_monitor_controls

`nsCamera.boards.LLNL_v4.llnl_v4.daedalus_monitor_controls`

Definition at line 291 of file [LLNL_v4.py](#).

6.8.4.5 daedalus_subreg_aliases

`nsCamera.boards.LLNL_v4.llnl_v4.daedalus_subreg_aliases`

Definition at line 264 of file [LLNL_v4.py](#).

6.8.4.6 defoff

`nsCamera.boards.LLNL_v4.llnl_v4.defoff`

Definition at line 207 of file [LLNL_v4.py](#).

6.8.4.7 defsens

`nsCamera.boards.LLNL_v4.llnl_v4.defsens`

Definition at line 208 of file [LLNL_v4.py](#).

6.8.4.8 icarus_monitor_controls

`nsCamera.boards.LLNL_v4.llnl_v4.icarus_monitor_controls`

Definition at line 252 of file [LLNL_v4.py](#).

6.8.4.9 icarus_subreg_aliases

`nsCamera.boards.LLNL_v4.llnl_v4.icarus_subreg_aliases`

Definition at line 213 of file [LLNL_v4.py](#).

6.8.4.10 logcrit

`nsCamera.boards.LLNL_v4.llnl_v4.logcrit`

Definition at line 183 of file [LLNL_v4.py](#).

6.8.4.11 logdebug

`nsCamera.boards.LLNL_v4.llnl_v4.logdebug`

Definition at line 187 of file [LLNL_v4.py](#).

6.8.4.12 logerr

`nsCamera.boards.LLNL_v4.llnl_v4.logerr`

Definition at line 184 of file [LLNL_v4.py](#).

6.8.4.13 loginfo

`nsCamera.boards.LLNL_v4.llnl_v4.loginfo`

Definition at line 186 of file [LLNL_v4.py](#).

6.8.4.14 logwarn

`nsCamera.boards.LLNL_v4.llnl_v4.logwarn`

Definition at line 185 of file [LLNL_v4.py](#).

6.8.4.15 registers

`nsCamera.boards.LLNL_v4.llnl_v4.registers` [static]

Definition at line 38 of file [LLNL_v4.py](#).

6.8.4.16 rs422_baud

`nsCamera.boards.LLNL_v4.llnl_v4.rs422_baud`

Definition at line 194 of file [LLNL_v4.py](#).

6.8.4.17 rs422_cmd_wait

`nsCamera.boards.LLNL_v4.llnl_v4.rs422_cmd_wait`

Definition at line 195 of file [LLNL_v4.py](#).

6.8.4.18 subregisters

`list nsCamera.boards.LLNL_v4.llnl_v4.subregisters` [static]

Definition at line 101 of file [LLNL_v4.py](#).

6.8.4.19 subreglist

`nsCamera.boards.LLNL_v4.llnl_v4.subreglist`

Definition at line 300 of file [LLNL_v4.py](#).

6.8.4.20 VREF

`nsCamera.boards.LLNL_v4.llnl_v4.VREF`

Definition at line 189 of file [LLNL_v4.py](#).

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

- [C:/Users/hill35/git/camera_python/nsCamera/boards/LLNL_v4.py](#)

6.9 nsCamera.utils.Packet.Packet Class Reference

Public Member Functions

- [__init__](#) (self, [preamble](#)="aaaa", [cmd](#)="0", [addr](#)="", [data](#)="00000000", [seqID](#)="", [payload_length](#)="", [payload](#)="", [crc](#)="")
- [pktStr](#) (self)
- [calculateCRC](#) (self)
- [checkCRC](#) (self)
- [checkReadPacket](#) (self, resppkt)
- [checkResponsePacket](#) (self, resppkt)
- [checkResponseString](#) (self, respstr)

Public Attributes

- [PY3](#)
- [preamble](#)
- [cmd](#)
- [addr](#)
- [data](#)
- [seqID](#)
- [payload_length](#)
- [payload](#)
- [crc](#)
- [type](#)

6.9.1 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 29 of file [Packet.py](#).

6.9.2 Constructor & Destructor Documentation

6.9.2.1 __init__()

```
nsCamera.utils.Packet.Packet.__init__ (
    self,
    preamble = "aaaa",
    cmd = "0",
    addr = "",
    data = "00000000",
    seqID = "",
    payload_length = "",
    payload = "",
    crc = "" )
```

Definition at line 52 of file [Packet.py](#).

```
00063 ):
00064     self.PY3 = sys.version_info > (3,)
00065     self.preamble = preamble # 16 bit packet preamble
00066     self.cmd = str(cmd) # 4 bit command packet
00067     self.addr = addr.zfill(3) # 12 bit address packet
00068     self.data = data.zfill(8) # 32 bit data packet
00069     # 16 bit sequence ID packet (only Read Burst)
00070     self.seqID = seqID
00071     # 16 bit payload packet (only Read Burst)
00072     self.payload_length = payload_length
00073     # variable payload packet (only Read Burst) for now it's 16 bits
00074     self.payload = payload
00075     # 16 bit CRC-CCIT (XModem) packet
00076     self.crc = crc
00077     self.type = ""
00078     if self.crc == "": # check if packet to be sent needs crc appended
00079         self.crc = self.calculateCRC()
00080
```

6.9.3 Member Function Documentation

6.9.3.1 calculateCRC()

```
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 108 of file [Packet.py](#).

```
00108 def calculateCRC(self):
00109     """
00110     Calculate CRC-CCIT (XModem) (2 bytes) from 8 byte packet for send and rcv
00111
00112     Returns:
00113         CRC as hexadecimal string without '0x'
00114     """
00115     preamble = self.preamble
00116     crc = self.crc
00117     self.crc = ""
00118     self.preamble = ""
00119
```

```

00120         CRC_dec = crc16pure.crc16xmodem(str2bytes(self.pktStr()))
00121         # input = int type decimal, output = hex string with 0x at the beginning
00122         CRC_hex_0x = "0x%0.4X" % CRC_dec
00123         # make all hex letters lower case for comparison
00124         CRC_hex = CRC_hex_0x.lower()
00125         # input = hex string with 0x at the beginning, output = hex str with 0x removed
00126         CRC_hex = CRC_hex[2:]
00127         self.preamble = preamble
00128         self.crc = crc
00129         return CRC_hex
00130

```

6.9.3.2 checkCRC()

```

nsCamera.utils.Packet.Packet.checkCRC (
    self )

```

Returns: boolean, True if CRC check passes

Definition at line 131 of file [Packet.py](#).

```

00131     def checkCRC(self):
00132         """
00133         Returns: boolean, True if CRC check passes
00134         """
00135         return self.calculateCRC() == self.crc
00136

```

6.9.3.3 checkReadPacket()

```

nsCamera.utils.Packet.Packet.checkReadPacket (
    self,
    resppkt )

```

Confirm that Read Single occurred without error

Args:

resp_pkt: response packet

Returns:

tuple (error string, response packet as string)

Definition at line 137 of file [Packet.py](#).

```

00137     def checkReadPacket(self, resp_pkt):
00138         """
00139         Confirm that Read Single occurred without error
00140         Args:
00141             resp_pkt: response packet
00142
00143         Returns:
00144             tuple (error string, response packet as string)
00145         """
00146         err = ""
00147         if int(resp_pkt.cmd.upper(), 16) - int(self.cmd.upper(), 16) != 0x8:
00148             err = "invalid command; "
00149         if resp_pkt.addr.upper() != self.addr.upper():
00150             err += "invalid address; "
00151         if resp_pkt.crc.upper() != resp_pkt.calculateCRC().upper():
00152             err += "invalid CRC; "
00153         return err, resp_pkt.pktStr()
00154

```

6.9.3.4 checkResponsePacket()

```
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 155 of file [Packet.py](#).

```
00155     def checkResponsePacket(self, resppkt):
00156         """
00157         Confirm that Write Single occurred without error
00158         Args:
00159             resppkt: response packet
00160
00161         Returns:
00162             tuple (error string, response packet as string)
00163         """
00164         err = ""
00165         if int(resppkt.data, 16) & 1:
00166             err += "Checksum error; "
00167         if int(resppkt.data, 16) & 2:
00168             err += "Invalid command / command not executed; "
00169         err1, rval = self.checkReadPacket(resppkt)
00170         err += err1
00171         return err, rval
00172
```

6.9.3.5 checkResponseString()

```
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 173 of file [Packet.py](#).

```
00173     def checkResponseString(self, respstr):
00174         """
00175         Checks response string for error indicators
00176         Args:
00177             respstr: packet as hexadecimal string
00178
00179         Returns:
00180             tuple (error string, response packet string)
00181         """
00182         respstring = respstr.decode(encoding="UTF-8")
00183         resppkt = Packet(
00184             preamble=respstring[0:4],
00185             cmd=respstring[4],
00186             addr=respstring[5:8],
00187             data=respstring[8:16],
00188         )
```

```

00189
00190     if resppkt.cmd == "8":
00191         # verify response to write command
00192         err, rval = self.checkResponsePacket(resppkt)
00193     elif resppkt.cmd == "9":
00194         err, rval = self.checkReadPacket(resppkt) # verify response to read command
00195     else:
00196         err = "Packet command invalid; "
00197         rval = ""
00198     return err, rval
00199
00200
00201 """
00202 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00203 LLNL-CODE-838080
00204
00205 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00206 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00207 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00208 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00209 be made under this license.
00210 """

```

6.9.3.6 pktStr()

```

nsCamera.utils.Packet.Packet.pktStr (
    self )

```

Generate hexadecimal string form of packet

Returns:

packet as hexadecimal string without '0x'

Definition at line 81 of file [Packet.py](#).

```

00081     def pktStr(self):
00082         """
00083         Generate hexadecimal string form of packet
00084
00085         Returns:
00086             packet as hexadecimal string without '0x'
00087         """
00088         if self.seqID != "":
00089             # Read burst response
00090             packetparts = [
00091                 self.preamble,
00092                 self.cmd,
00093                 self.seqID,
00094                 self.payload_length,
00095                 self.payload,
00096                 self.crc,
00097             ]
00098         else:
00099             # Single Command/Response response
00100             packetparts = [self.preamble, self.cmd, self.addr, self.data, self.crc]
00101             stringparts = [
00102                 part.decode("ascii") if isinstance(part, bytes) else part
00103                 for part in packetparts
00104             ]
00105             out = "".join(stringparts)
00106             return out
00107

```

6.9.4 Member Data Documentation

6.9.4.1 addr

```

nsCamera.utils.Packet.Packet.addr

```

Definition at line 67 of file [Packet.py](#).

6.9.4.2 cmd

`nsCamera.utils.Packet.Packet.cmd`

Definition at line 66 of file [Packet.py](#).

6.9.4.3 crc

`nsCamera.utils.Packet.Packet.crc`

Definition at line 76 of file [Packet.py](#).

6.9.4.4 data

`nsCamera.utils.Packet.Packet.data`

Definition at line 68 of file [Packet.py](#).

6.9.4.5 payload

`nsCamera.utils.Packet.Packet.payload`

Definition at line 74 of file [Packet.py](#).

6.9.4.6 payload_length

`nsCamera.utils.Packet.Packet.payload_length`

Definition at line 72 of file [Packet.py](#).

6.9.4.7 preamble

`nsCamera.utils.Packet.Packet.preamble`

Definition at line 65 of file [Packet.py](#).

6.9.4.8 PY3

`nsCamera.utils.Packet.Packet.PY3`

Definition at line 64 of file [Packet.py](#).

6.9.4.9 seqID

`nsCamera.utils.Packet.Packet.seqID`

Definition at line 70 of file [Packet.py](#).

6.9.4.10 type

`nsCamera.utils.Packet.Packet.type`

Definition at line 77 of file [Packet.py](#).

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

- C:/Users/hill35/git/camera_python/nsCamera/utils/[Packet.py](#)

6.10 nsCamera.comms.RS422.RS422 Class Reference

Public Member Functions

- [__init__](#) (self, camassem, [baud](#)=921600, [par](#)="O", [stop](#)=1)
- [serialClose](#) (self)
- [sendCMD](#) (self, pkt)
- [arm](#) (self, [mode](#))
- [readFrames](#) (self, waitOnSRAM, timeout=0, fast=False, columns=1)
- [readoff](#) (self, waitOnSRAM, timeout, fast, columns=1)
- [writeSerial](#) (self, outstring, timeout)
- [readSerial](#) (self, size, timeout=None)
- [closeDevice](#) (self)

Public Attributes

- [ca](#)
- [logcrit](#)
- [logerr](#)
- [logwarn](#)
- [loginfo](#)
- [logdebug](#)
- [mode](#)
- [baud](#)
- [par](#)
- [stop](#)
- [read_timeout](#)
- [write_timeout](#)
- [datatimeout](#)
- [PY3](#)
- [skipError](#)
- [port](#)
- [payloadsize](#)

Protected Attributes

- `_ser`

6.10.1 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
readFrames() - waits for data ready register flag, then copies camera image data
               into numpy arrays
readoff() - waits for data ready register flag, then copies camera image data
            into numpy arrays; returns payload, payload size, and error message
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 30 of file [RS422.py](#).

6.10.2 Constructor & Destructor Documentation

6.10.2.1 `__init__()`

```
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 48 of file [RS422.py](#).

```
00048     def __init__(self, camassem, baud=921600, par="O", stop=1):
00049         """
00050         Args:
00051             camassem: parent cameraAssembler object
00052             baud: bits per second
00053             par: parity type
00054             stop: number of stop bits
00055         """
00056         self.ca = camassem
00057         self.logcrit = self.ca.logcritbase + "[RS422] "
00058         self.logerr = self.ca.logerrbase + "[RS422] "
00059         self.logwarn = self.ca.logwarnbase + "[RS422] "
00060         self.loginfo = self.ca.loginfobase + "[RS422] "
00061         self.logdebug = self.ca.logdebugbase + "[RS422] "
00062         logging.info(self.loginfo + "initializing RS422 comms object")
00063         logging.debug(
```

```

00064         self.logdebug
00065         + "Init: baud = "
00066         + str(baud)
00067         + "; par = "
00068         + str(par)
00069         + "; stop = "
00070         + str(stop)
00071     )
00072     self.mode = 0
00073     self.baud = baud # Baud rate (bits/second)
00074     self.par = par # Parity bit
00075     self.stop = stop # Number of stop bits
00076     self.read_timeout = 1 # default timeout for ordinary packets
00077     self.write_timeout = 1
00078     # TODO: make datatimeout a cameraAssembler parameter
00079     self.datatimeout = 60 # timeout for data read
00080     logging.debug(
00081         self.logdebug + "Data timeout = " + str(self.datatimeout) + " seconds"
00082     )
00083     self.PY3 = sys.version_info > (3,)
00084     self.skipError = False
00085     port = ""
00086     ports = list(serial.tools.list_ports.comports())
00087     logging.debug(self.logdebug + "Comports: " + str(ports))
00088     for p, desc, add in ports:
00089         if self.ca.port is None or p == "COM" + str(self.ca.port):
00090             logging.info(self.loginfo + "found comm port " + p)
00091             try:
00092                 with serial.Serial(
00093                     p,
00094                     self.baud,
00095                     parity=self.par,
00096                     timeout=0.01,
00097                     write_timeout=0.01,
00098                 ) as ser:
00099                     ser.write(str2bytes("aaaa10000000000001a84"))
00100                     time.sleep(1)
00101                     s = ser.read(10)
00102                     resp = bytes2str(s)
00103                     logging.debug(self.logdebug + "Init response: " + str(resp))
00104                     if (
00105                         resp[0:5].lower() == "aaaa9"
00106                     ): # TODO: add check for RS422 bit in board description
00107                         boardid = resp[8:10]
00108                         if boardid == "00":
00109                             logging.critical(
00110                                 self.logcrit + "SNLrevC board detected - not "
00111                                 "compatible with nsCamera >= 2.0"
00112                             )
00113                             sys.exit(1)
00114                         elif boardid == "81":
00115                             logging.info(self.loginfo + "LLNLv1 board detected")
00116                         elif boardid == "84":
00117                             logging.info(self.loginfo + "LLNLv4 board detected")
00118                         else:
00119                             logging.info(
00120                                 self.loginfo + "unidentified board detected"
00121                             )
00122                         logging.info(self.loginfo + "connected to " + p)
00123                         port = p
00124                         ser.reset_input_buffer()
00125                         ser.reset_output_buffer()
00126                         break
00127             except Exception as e:
00128                 logging.error(self.logerr + "port identification: " + str(e))
00129     if port == "":
00130         if self.ca.port:
00131             logging.critical(
00132                 self.logcrit + "No usable board found at port " + str(self.ca.port)
00133             )
00134             sys.exit(1)
00135         else:
00136             logging.critical(self.logcrit + "No usable board found")
00137             sys.exit(1)
00138     self.port = port # COM port to use for RS422 link
00139     self.ca.port = port[3:] # re-extract port number from com name
00140
00141     self._ser = serial.Serial( # Class RS422
00142         port=self.port,
00143         baudrate=self.baud,
00144         parity=self.par,

```



```

00145         stopbits=self.stop,
00146         timeout=self.read_timeout, # timeout for serial read
00147         bytesize=serial.EIGHTBITS,
00148     )
00149     self.payloadsize = (
00150         self.ca.sensor.width
00151         * self.ca.sensor.height
00152         * self.ca.sensor.nframes
00153         * self.ca.sensor.bytesperpixel
00154     )
00155     logging.debug(
00156         self.logdebug + "Payload size: " + str(self.payloadsize) + " bytes"
00157     )
00158     self._ser.flushInput()
00159     if not self._ser.is_open:
00160         logging.critical(self.logcrit + "Unable to open serial connection")
00161         sys.exit(1)
00162

```

6.10.3 Member Function Documentation

6.10.3.1 arm()

```

nsCamera.comms.RS422.RS422.arm (
    self,
    mode )

```

Puts camera into wait state for trigger. Mode determines source; defaults to 'Hardware'

Args:

```

mode:  'Software'|'S' activates software, disables hardware triggering
       'Hardware'|'H' activates hardware, disables software triggering
       Hardware is the default

```

Returns:

```

tuple (error, response string)

```

Definition at line 319 of file [RS422.py](#).

```

00319     def arm(self, mode):
00320         """
00321         Puts camera into wait state for trigger. Mode determines source; defaults to
00322         'Hardware'
00323
00324         Args:
00325             mode:  'Software'|'S' activates software, disables hardware triggering
00326                   'Hardware'|'H' activates hardware, disables software triggering
00327                   Hardware is the default
00328
00329         Returns:
00330             tuple (error, response string)
00331         """
00332         if not mode:
00333             mode = "Hardware"
00334         logging.info(self.loginfo + "arm")
00335         logging.debug(self.logdebug + "arming mode: " + str(mode))
00336         self.ca.clearStatus()
00337         self.ca.latchPots()
00338         err, resp = self.ca.startCapture(mode)
00339         if err:
00340             logging.error(self.logerr + "unable to arm camera")
00341         else:
00342             self.ca.armed = True
00343             self.skipError = True
00344         return err, resp
00345

```

6.10.3.2 closeDevice()

```
nsCamera.comms.RS422.RS422.closeDevice (
    self )
```

Close primary serial interface

Definition at line 496 of file [RS422.py](#).

```
00496     def closeDevice(self):
00497         """
00498         Close primary serial interface
00499         """
00500         logging.debug(self.logdebug + "Closing RS422 connection")
00501         self._ser.close()
00502
00503
00504     """
00505     Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00506     LLNL-CODE-838080
00507
00508     This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00509     contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00510     and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00511     'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00512     be made under this license.
00513     """
```

6.10.3.3 readFrames()

```
nsCamera.comms.RS422.RS422.readFrames (
    self,
    waitOnSRAM,
    timeout = 0,
    fast = False,
    columns = 1 )
```

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
columns: 1 for single image per frame, 2 for separate hemisphere images
```

Returns:

```
list of numpy arrays OR raw text stream
```

Definition at line 346 of file [RS422.py](#).

```
00346     def readFrames(self, waitOnSRAM, timeout=0, fast=False, columns=1):
00347         """
00348         Copies image data from board into numpy arrays.
00349
00350         Args:
00351             waitOnSRAM: if True, wait until SRAM_READY flag is asserted to begin copying
00352             data
00353             timeout: passed to waitForSRAM; after this many seconds begin copying data
```

```

00354             irrespective of SRAM_READY status; 'zero' means wait indefinitely
00355             WARNING: If acquisition fails, the SRAM will not contain a current image,
00356             but the code will copy the data anyway
00357             fast: if False, parse and convert frames to numpy arrays; if True, return
00358             unprocessed text stream
00359             columns: 1 for single image per frame, 2 for separate hemisphere images
00360
00361         Returns:
00362             list of numpy arrays OR raw text stream
00363
00364         """
00365         frames, _, _ = self.readoff(waitOnSRAM, timeout, fast, columns)
00366         return frames
00367

```

6.10.3.4 readoff()

```

nsCamera.comms.RS422.RS422.readoff (
    self,
    waitOnSRAM,
    timeout,
    fast,
    columns = 1 )

```

Copies image data from board into numpy arrays; returns data, length of data, and error messages. Use 'readFrames()' unless you require this additional information

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
 columns: 1 for single image per frame, 2 for separate hemisphere images

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 368 of file RS422.py.

```

00368     def readoff(self, waitOnSRAM, timeout, fast, columns=1):
00369         """
00370         Copies image data from board into numpy arrays; returns data, length of data,
00371         and error messages. Use 'readFrames()' unless you require this additional
00372         information
00373
00374         Args:
00375             waitOnSRAM: if True, wait until SRAM_READY flag is asserted to begin copying
00376             data
00377             timeout: passed to waitForSRAM; after this many seconds begin copying data
00378             irrespective of SRAM_READY status; 'zero' means wait indefinitely
00379             WARNING: If acquisition fails, the SRAM will not contain a current image,
00380             but the code will copy the data anyway
00381             fast: if False, parse and convert frames to numpy arrays; if True, return
00382             unprocessed text stream
00383             columns: 1 for single image per frame, 2 for separate hemisphere images
00384
00385         Returns:
00386             tuple (list of numpy arrays OR raw text stream, length of downloaded payload
00387             in bytes, payload error flag)

```

```

00388         NOTE: This reduces readoff by <1 second, so will have no noticeable impact
00389             when using RS422
00390         """
00391         logging.info(self.loginfo + "readoff")
00392         logging.debug(
00393             self.logdebug
00394             + "readoff: waitonSRAM = "
00395             + str(waitOnSRAM)
00396             + "; timeout = "
00397             + str(timeout)
00398             + "; fast = "
00399             + str(fast)
00400         )
00401         errortemp = False
00402
00403         # Wait for data to be ready on board, turns off error messaging
00404         # Skip wait only if explicitly tagged 'False' ('None' defaults to True)
00405         if waitOnSRAM is not False:
00406             logging.getLogger().setLevel(logging.CRITICAL)
00407             self.ca.waitForSRAM(timeout)
00408             logging.getLogger().setLevel(self.ca.verblevel)
00409
00410         # Retrieve data
00411         err, rval = self.ca.readSRAM()
00412         if err:
00413             logging.error(self.logerr + "Error detected in readSRAM")
00414             time.sleep(0.3)
00415             logging.debug(self.logdebug + "readoff: first 64 chars: " + str(rval[0:64]))
00416             # extract only the read burst data. Remove header & CRC footer
00417             read_burst_data = rval[36:-4]
00418
00419         # Payload size as string implied by provided parameters
00420         expectedlength = (
00421             4
00422             * (self.ca.sensor.lastframe - self.ca.sensor.firstframe + 1)
00423             * (self.ca.sensor.lastrow - self.ca.sensor.firstrow + 1)
00424             * self.ca.sensor.width
00425         )
00426         padding = expectedlength - len(read_burst_data)
00427         if padding:
00428             logging.warning(
00429                 "{logwarn}readoff: Payload is shorter than expected."
00430                 " Padding with '0's".format(logwarn=self.logwarn)
00431             )
00432             read_burst_data = read_burst_data.ljust(expectedlength, "0")
00433
00434         if fast:
00435             return read_burst_data, len(read_burst_data) // 2, errortemp
00436         else:
00437             parsed = generateFrames(self.ca, read_burst_data, columns)
00438             return parsed, len(read_burst_data) // 2, errortemp
00439

```

6.10.3.5 readSerial()

```

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 465 of file [RS422.py](#).

```

00465     def readSerial(self, size, timeout=None):
00466         """
00467         Read bytes from the serial port. Does not verify packets.
00468
00469         Args:
00470             size: number of bytes to read
00471             timeout: serial timeout in sec
00472
00473         Returns:
00474             tuple (error string, string read from serial port)
00475         """
00476         logging.debug(
00477             self.logdebug
00478             + "readSerial: size = "
00479             + str(size)
00480             + "; timeout = "
00481             + str(timeout)
00482         )
00483         err = ""
00484         if timeout:
00485             self._ser.timeout = timeout
00486         else:
00487             self._ser.timeout = self.read_timeout
00488         resp = self._ser.read(size)
00489         if len(resp) < 10: # bytes
00490             err += (
00491                 self.logerr + "readSerial : packet too small: '" + bytes2str(resp) + "'"
00492             )
00493             logging.error(err)
00494         return err, bytes2str(resp)
00495

```

6.10.3.6 sendCMD()

```

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 170 of file [RS422.py](#).

```

00170     def sendCMD(self, pkt):
00171         """
00172         Submit packet and verify response packet. Recognizes readoff packet and adjusts
00173         read size and timeout appropriately
00174
00175         Args:
00176             pkt: Packet object
00177
00178         Returns:
00179             tuple (error, response string)
00180         """
00181         pktStr = pkt.pktStr()
00182         logging.debug(self.logdebug + "sendCMD packet: " + str(pktStr))
00183         self._ser.flushInput()
00184         time.sleep(0.01) # wait 10 ms in between flushing input and output buffers
00185         self._ser.flushOutput()
00186         self.ca.writeSerial(pktStr)
00187         err0 = ""
00188         err = ""
00189         resp = ""
00190         tries = 3 # TODO: make a function parameter?
00191

```

```

00192         if (
00193             hasattr(self.ca, "board")
00194             and pktStr[4] == "0"
00195             and pktStr[5:8] == self.ca.board.registers["SRAM_CTL"]
00196         ):
00197             # download data payload
00198             logging.info(
00199                 self.loginfo + "Payload size (bytes) = " + str(self.payloadsize)
00200             )
00201             crcresp0 = ""
00202             crcresp1 = ""
00203             smallresp = ""
00204             emptyResponse = False
00205             wrongSize = False
00206             # TODO: refactor payload error management to another method
00207             for i in range(tries):
00208                 err, resp = self.readSerial(
00209                     self.payloadsize + 20, timeout=self.datatimeout
00210                 )
00211                 if err:
00212                     logging.error(
00213                         self.logerr + "sendCMD: read payload failed " + pktStr + err
00214                     )
00215                     self.ca.payloaderror = True
00216                 else:
00217                     if not len(resp):
00218                         err0 = self.logerr + "sendCMD: empty response from board"
00219                         logging.error(err0)
00220                         emptyResponse = True
00221                         self.ca.payloaderror = True
00222                     elif len(resp) != 2 * (self.payloadsize + 20):
00223                         err0 = (
00224                             self.logerr
00225                             + "sendCMD: incorrect response; expected "
00226                             + str(self.payloadsize + 20)
00227                             + " bytes, received "
00228                             + str(len(resp) // 2)
00229                         )
00230                         logging.error(err0)
00231                         wrongSize = True
00232                         smallresp = resp
00233                         self.ca.payloaderror = True
00234                     elif not checkCRC(resp[4:20]):
00235                         err0 = (
00236                             self.logerr
00237                             + "sendCMD: "
00238                             + pktStr
00239                             + " - payload preface CRC fail"
00240                         )
00241                         logging.error(err0)
00242                         self.ca.payloaderror = True
00243                         crcresp1 = resp
00244                     elif not checkCRC(resp[24:]):
00245                         err0 = (
00246                             self.logerr + "sendCMD: " + pktStr + " - payload CRC fail"
00247                         )
00248                         logging.error(err0)
00249                         self.ca.payloaderror = True
00250                         crcresp0 = resp
00251                 err += err0
00252             time.sleep(5)
00253             if self.ca.payloaderror:
00254                 # keep best results over multiple tries; e.g., if first try is
00255                 # bad CRC and second try is an incomplete payload, use the
00256                 # first payload
00257                 if i == tries - 1:
00258                     if crcresp0:
00259                         logging.error(
00260                             self.logerr + "sendCMD: Unable to acquire "
00261                             "CRC-confirmed payload after "
00262                             + str(tries)
00263                             + " attempts. Continuing with unconfirmed payload"
00264                         )
00265                     resp = crcresp0
00266                 elif crcresp1:
00267                     logging.error(
00268                         self.logerr + "sendCMD: Unable to acquire "
00269                         "CRC-confirmed readoff after "
00270                         + str(tries)
00271                         + " attempts. Continuing with unconfirmed payload"
00272                     )

```

```

00273         resp = crcresp1
00274     elif wrongSize:
00275         logging.error(
00276             self.logerr + "sendCMD: Unable to acquire complete "
00277             "payload after "
00278             + str(tries)
00279             + " attempts. Dumping datastream to file."
00280         )
00281         resp = smallresp
00282         self.ca.dumpNumpy(resp)
00283     elif emptyResponse:
00284         logging.error(
00285             self.logerr + "sendCMD: Unable to acquire any "
00286             "payload after " + str(tries) + " attempts."
00287         )
00288     else:
00289         logging.info(
00290             self.loginfo + "Retrying download, attempt #" + str(i + 1)
00291         )
00292         err = ""
00293         err0 = ""
00294         self.ca.payloaderror = False
00295         self.ca.writeSerial(pktStr)
00296     else:
00297         logging.info(self.loginfo + "Download successful")
00298         if self.ca.boardname == "llnl_v4":
00299             # self.ca.setSubregister('SWACK','1')
00300             pass
00301             break
00302
00303     else:
00304         # non-payload messages and workaround for initial setup before board object
00305         # has been initialized
00306         time.sleep(0.03)
00307         self._ser.timeout = 0.02
00308         err, resp = self.readSerial(10)
00309         logging.debug(self.logdebug + "sendCMD response: " + str(resp))
00310         if err:
00311             logging.error(
00312                 self.logerr + "sendCMD: readSerial failed (regular packet) " + err
00313             )
00314         elif not checkCRC(resp[4:20]):
00315             err = self.logerr + "sendCMD- regular packet CRC fail: " + resp
00316             logging.error(err)
00317     return err, resp
00318

```

6.10.3.7 serialClose()

```

nsCamera.comms.RS422.RS422.serialClose (
    self )

```

Close serial interface

Definition at line 163 of file [RS422.py](#).

```

00163     def serialClose(self):
00164         """
00165         Close serial interface
00166         """
00167         logging.debug(self.logdebug + "serialclose")
00168         self._ser.close() # close serial interface COM port
00169

```

6.10.3.8 writeSerial()

```

nsCamera.comms.RS422.RS422.writeSerial (
    self,
    outstring,
    timeout )

```

Transmit string to board

Args:

 outstring: string to write
 timeout: serial timeout in sec

Returns:

 integer length of string written to serial port

Definition at line 440 of file [RS422.py](#).

```
00440     def writeSerial(self, outstring, timeout):
00441         """
00442             Transmit string to board
00443
00444             Args:
00445                 outstring: string to write
00446                 timeout: serial timeout in sec
00447             Returns:
00448                 integer length of string written to serial port
00449             """
00450         logging.debug(
00451             self.logdebug
00452             + "writeSerial: outstring = "
00453             + str(outstring)
00454             + "; timeout = "
00455             + str(timeout)
00456         )
00457         if timeout:
00458             self._ser.timeout = timeout
00459         else:
00460             self._ser.timeout = self.write_timeout
00461         lengthwritten = self._ser.write(str2bytes(outstring))
00462         self._ser.timeout = self.read_timeout # reset if changed above
00463         return lengthwritten
00464
```

6.10.4 Member Data Documentation

6.10.4.1 `_ser`

`nsCamera.comms.RS422.RS422._ser` [protected]

Definition at line 141 of file [RS422.py](#).

6.10.4.2 `baud`

`nsCamera.comms.RS422.RS422.baud`

Definition at line 73 of file [RS422.py](#).

6.10.4.3 `ca`

`nsCamera.comms.RS422.RS422.ca`

Definition at line 56 of file [RS422.py](#).

6.10.4.4 datatimeout

`nsCamera.comms.RS422.RS422.datatimeout`

Definition at line 79 of file [RS422.py](#).

6.10.4.5 logcrit

`nsCamera.comms.RS422.RS422.logcrit`

Definition at line 57 of file [RS422.py](#).

6.10.4.6 logdebug

`nsCamera.comms.RS422.RS422.logdebug`

Definition at line 61 of file [RS422.py](#).

6.10.4.7 logerr

`nsCamera.comms.RS422.RS422.logerr`

Definition at line 58 of file [RS422.py](#).

6.10.4.8 loginfo

`nsCamera.comms.RS422.RS422.loginfo`

Definition at line 60 of file [RS422.py](#).

6.10.4.9 logwarn

`nsCamera.comms.RS422.RS422.logwarn`

Definition at line 59 of file [RS422.py](#).

6.10.4.10 mode

`nsCamera.comms.RS422.RS422.mode`

Definition at line 72 of file [RS422.py](#).

6.10.4.11 par

`nsCamera.comms.RS422.RS422.par`

Definition at line 74 of file [RS422.py](#).

6.10.4.12 payloadsize

`nsCamera.comms.RS422.RS422.payloadsize`

Definition at line 149 of file [RS422.py](#).

6.10.4.13 port

`nsCamera.comms.RS422.RS422.port`

Definition at line 138 of file [RS422.py](#).

6.10.4.14 PY3

`nsCamera.comms.RS422.RS422.PY3`

Definition at line 83 of file [RS422.py](#).

6.10.4.15 read_timeout

`nsCamera.comms.RS422.RS422.read_timeout`

Definition at line 76 of file [RS422.py](#).

6.10.4.16 skipError

`nsCamera.comms.RS422.RS422.skipError`

Definition at line 84 of file [RS422.py](#).

6.10.4.17 stop

`nsCamera.comms.RS422.RS422.stop`

Definition at line 75 of file [RS422.py](#).

6.10.4.18 write_timeout

nsCamera.comms.RS422.RS422.write_timeout

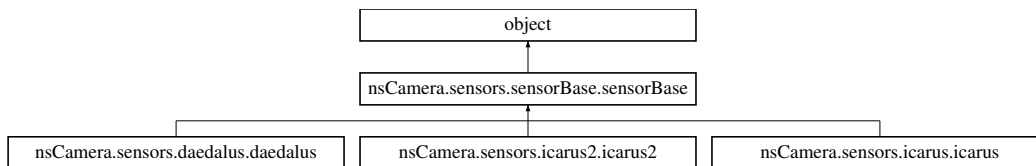
Definition at line 77 of file [RS422.py](#).

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

- C:/Users/hill35/git/camera_python/nsCamera/comms/[RS422.py](#)

6.11 nsCamera.sensors.sensorBase.sensorBase Class Reference

Inheritance diagram for nsCamera.sensors.sensorBase.sensorBase:



Public Member Functions

- `__init__` (self, camassem)
- `init_board_specific` (self)
- `checkSensorVoltStat` (self)
- `setInterlacing` (self, ifactor)
- `setHighFullWell` (self, flag)
- `setZeroDeadTime` (self, flag)
- `setTriggerDelay` (self, delay)
- `setPhiDelay` (self, delay)
- `setExtClk` (self, delay)
- `setTiming` (self, side="AB", sequence=None, delay=0)
- `setArbTiming` (self, side="AB", sequence=None)
- `getTiming` (self, side, actual)
- `setManualShutters` (self, timing=None)
- `setManualTiming` (self, timing=None)
- `getManualTiming` (self)
- `getSensTemp` (self, scale=None, offset=None, slope=None, dec=None)
- `selectOscillator` (self, osc=None)
- `parseReadoff` (self, frames, columns)
- `getSensorStatus` (self)
- `reportStatusSensor` (self, statusbits, statusbits2)

Public Attributes

- [ca](#)
- [logcrit](#)
- [logerr](#)
- [logwarn](#)
- [loginfo](#)
- [logdebug](#)
- [fpganumID](#)
- [sensfam](#)

6.11.1 Detailed Description

Base class for sensors. 'Virtual' methods below default to Icarus behavior. daedalus.py overrides these methods as necessary

Definition at line 25 of file [sensorBase.py](#).

6.11.2 Constructor & Destructor Documentation

6.11.2.1 `__init__()`

```
nsCamera.sensors.sensorBase.sensorBase.__init__ (
    self,
    camassem )
```

Reimplemented in [nsCamera.sensors.daedalus.daedalus](#), [nsCamera.sensors.icarus.icarus](#), and [nsCamera.sensors.icarus2.icarus2](#).

Definition at line 31 of file [sensorBase.py](#).

```
00031     def __init__(self, camassem):
00032         self.ca = camassem
00033         # skip board settings if no board object exists
00034         if hasattr(self.ca, "board"):
00035             self.init_board_specific()
00036
00037         (
00038             self.logcrit,
00039             self.logerr,
00040             self.logwarn,
00041             self.loginfo,
00042             self.logdebug,
00043         ) = makeLogLabels(self.ca.logtag, self.loglabel)
00044
00045         # skip assignment if no comms object exists
00046         if hasattr(self.ca, "comms"):
00047             self.ca.comms.payloadsize = (
00048                 self.width * self.height * self.nframes * self.bytesperpixel
00049             )
00050
00051         logging.info(self.loginfo + "Initializing sensor object")
00052
```

6.11.3 Member Function Documentation

6.11.3.1 checkSensorVoltStat()

```
nsCamera.sensors.sensorBase.sensorBase.checkSensorVoltStat (
    self )
```

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

Returns:
boolean, True if jumpers select for Icarus sensor

Reimplemented in [nsCamera.sensors.icarus.icarus](#).

Definition at line 64 of file [sensorBase.py](#).

```
00064     def checkSensorVoltStat(self):
00065         """
00066         Checks register tied to sensor select jumpers to confirm match with sensor
00067         object
00068
00069         Returns:
00070             boolean, True if jumpers select for Icarus sensor
00071         """
00072         logging.debug(self.logdebug + "checkSensorVoltStat")
00073         err, status = self.ca.getSubregister(self.detect)
00074         if err:
00075             logging.error(self.logerr + "Unable to confirm sensor status")
00076             return False
00077         if not int(status):
00078             logging.error(self.logerr + self.sensfam + " sensor not detected")
00079             return False
00080         return True
00081
```

6.11.3.2 getManualTiming()

```
nsCamera.sensors.sensorBase.sensorBase.getManualTiming (
    self )
```

Read off manual shutter timing settings

Overridden in [daedalus.py](#)

Returns:
list of 2 lists of timing from A and B sides, respectively

Reimplemented in [nsCamera.sensors.daedalus.daedalus](#).

Definition at line 611 of file [sensorBase.py](#).

```
00611     def getManualTiming(self):
00612         """
00613         Read off manual shutter timing settings
00614         Overridden in daedalus.py
00615         Returns:
00616             list of 2 lists of timing from A and B sides, respectively
00617         """
00618         aside = []
00619         bside = []
00620         for reg in [
00621             "W0_INTEGRATION",
00622             "W0_INTERFRAME",
00623             "W1_INTEGRATION",
```

```

00624         "W1_INTERFRAME",
00625         "W2_INTEGRATION",
00626         "W2_INTERFRAME",
00627         "W3_INTEGRATION",
00628     ]:
00629         _, reghex = self.ca.getRegister(reg)
00630         aside.append(25 * int(reghex, 16))
00631     for reg in [
00632         "W0_INTEGRATION_B",
00633         "W0_INTERFRAME_B",
00634         "W1_INTEGRATION_B",
00635         "W1_INTERFRAME_B",
00636         "W2_INTEGRATION_B",
00637         "W2_INTERFRAME_B",
00638         "W3_INTEGRATION_B",
00639     ]:
00640         _, reghex = self.ca.getRegister(reg)
00641         bside.append(25 * int(reghex, 16))
00642     return [aside, bside]
00643

```

6.11.3.3 getSensorStatus()

```

nsCamera.sensors.sensorBase.sensorBase.getSensorStatus (
    self )

```

Wrapper for reportSensorStatus so that the user doesn't have to query statusbits

Definition at line 691 of file [sensorBase.py](#).

```

00691     def getSensorStatus(self):
00692         """
00693         Wrapper for reportSensorStatus so that the user doesn't have to query statusbits
00694         """
00695         sb1 = self.ca.board.checkstatus()
00696         sb2 = self.ca.board.checkstatus2()
00697         self.reportStatusSensor(sb1, sb2)
00698

```

6.11.3.4 getSensTemp()

```

nsCamera.sensors.sensorBase.sensorBase.getSensTemp (
    self,
    scale = None,
    offset = None,
    slope = None,
    dec = None )

```

Virtual method (Temperature sensor is not present on Icarus sensors). Returns 0.
Overridden by Daedalus method

Reimplemented in [nsCamera.sensors.daedalus.daedalus](#).

Definition at line 644 of file [sensorBase.py](#).

```

00644     def getSensTemp(self, scale=None, offset=None, slope=None, dec=None):
00645         """
00646         Virtual method (Temperature sensor is not present on Icarus sensors). Returns 0.
00647         Overridden by Daedalus method
00648         """
00649         return 0
00650

```

6.11.3.5 getTiming()

```
nsCamera.sensors.sensorBase.sensorBase.getTiming (
    self,
    side,
    actual )
```

actual = True: returns actual high speed intervals that will be generated by the FPGA as list
 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= True: list of shutter intervals;
 icarus: [delay, open0, closed0, open1, closed1, open2, closed2, open3]
 daedalus: [delay, open0, closed0, open1, closed1, open2]
 False: tuple (hemisphere label,
 'open shutter' in ns,
 'closed shutter' in ns,
 initial delay in ns)

Definition at line 434 of file [sensorBase.py](#).

```
00434     def getTiming(self, side, actual):
00435         """
00436         actual = True: returns actual high speed intervals that will be generated by the
00437             FPGA as list
00438             False: Returns high speed timing settings as set by setTiming. Assumes
00439                 that timing was set via the setTiming method--it will not accurately
00440                 report arbitrary timings set by direct register sets or manual
00441                 shutter control
00442
00443         Args:
00444             side: Hemisphere 'A' or 'B'
00445             actual: False: return HST settings
00446                   True: calculate and return actual HST behavior
00447
00448         Returns:
00449             actual= True: list of shutter intervals;
00450                   icarus: [delay, open0, closed0, open1, closed1, open2, closed2,
00451                           open3]
00452                   daedalus: [delay, open0, closed0, open1, closed1, open2]
00453             False: tuple (hemisphere label,
00454                   'open shutter' in ns,
00455                   'closed shutter' in ns,
00456                   initial delay in ns)
00457
00458         """
00459         logging.info("{}getTiming".format(self.loginf))
00460         if side is None:
00461             side = "A"
00462         logging.info(self.loginf + "get timing, side " + side.upper())
00463         if side.upper() == "A":
00464             lowreg = "HS_TIMING_DATA_ALO"
00465             highreg = "HS_TIMING_DATA_AHI"
00466         elif side.upper() == "B":
00467             lowreg = "HS_TIMING_DATA_BLO"
00468             highreg = "HS_TIMING_DATA_BHI"
00469         else:
00470             logging.error(
00471                 self.logerr
00472                 + "Invalid sensor side: "
00473                 + side
```

```

00474         + "; timing settings unchanged"
00475     )
00476     return "", 0, 0, 0
00477     err, lowpart = self.ca.getRegister(lowreg)
00478     err1, highpart = self.ca.getRegister(highreg)
00479     if err or err1:
00480         logging.error(
00481             self.logerr + "Unable to retrieve timing setting (getTiming), "
00482             "returning zeroes "
00483         )
00484         return side.upper(), 0, 0, 0
00485     full40hex = highpart[-2:] + lowpart.zfill(8)
00486     full40bin = "{0:0=40b}".format(int(full40hex, 16))
00487     logging.debug(self.logdebug + "full40bin = " + str(full40bin))
00488     if actual:
00489         if full40bin == "0" * 40: # all-zero timing
00490             if self.fpganumID == 2:
00491                 times = [0] * 6
00492             else:
00493                 times = [0] * 8
00494         else:
00495             full160 = 4 * full40bin
00496             gblist = [[k, len(list(g))] for k, g in itertools.groupby(full160)]
00497             if self.fpganumID == 2:
00498                 times = [int(x[1]) for x in gblist[:-7:-1]]
00499             else:
00500                 times = [int(x[1]) for x in gblist[:-9:-1]]
00501             times[0] = times[0] - 1
00502         if self.ca.sensorname == "icarus":
00503             # get timing for frames 1 and 2, keep delay as offset
00504             # TODO: should this give a 'correct' offset from frame 0?
00505             times12 = [times[0]] + times[3:6]
00506             return times12
00507         return times
00508     else:
00509         if full40bin == "0" * 40: # all-zero timing
00510             timeon, timeoff, delay = (0, 0, 0)
00511         else:
00512             gblist = [[k, len(list(g))] for k, g in itertools.groupby(full40bin)]
00513             delay = gblist[-1][1] - 1
00514             timeon = gblist[-2][1]
00515
00516             if self.ca.sensorname == "icarus":
00517                 if len(gblist) == 2: # 39,1 corner case
00518                     timeoff = 1
00519                 elif len(gblist) == 3: # sequence fits only once
00520                     timeoff = 40 - timeon
00521             else:
00522                 timeoff = gblist[-3][1]
00523         else:
00524             if len(gblist) < self.nframes: # sequence fits only once
00525                 timeoff = 40 - timeon
00526             else:
00527                 # TODO: confirm '-3' works for daedalus
00528                 timeoff = gblist[-3][1]
00529         return side.upper(), timeon, timeoff, delay
00530

```

6.11.3.6 init_board_specific()

```

nsCamera.sensors.sensorBase.sensorBase.init_board_specific (
    self )

```

Initialize aliases and subregisters specific to the current board and sensor.

Definition at line 53 of file [sensorBase.py](#).

```

00053     def init_board_specific(self):
00054         """Initialize aliases and subregisters specific to the current board and sensor."""
00055
00056         if self.ca.sensorname == "icarus" or self.ca.sensorname == "icarus2":
00057             self.ca.board.subreg_aliases = self.ca.board.icarus_subreg_aliases
00058             self.ca.board.monitor_controls = self.ca.board.icarus_monitor_controls
00059         else:
00060             self.ca.board.subreg_aliases = self.ca.board.daedalus_subreg_aliases
00061             self.ca.board.monitor_controls = self.ca.board.daedalus_monitor_controls
00062

```


6.11.3.7 parseReadoff()

```
nsCamera.sensors.sensorBase.sensorBase.parseReadoff (
    self,
    frames,
    columns )
```

Virtual method (Order parsing is unnecessary for Icarus, continue to hemisphere parsing.)
Overridden by Daedalus method

Reimplemented in [nsCamera.sensors.daedalus.daedalus](#).

Definition at line 683 of file [sensorBase.py](#).

```
00683     def parseReadoff(self, frames, columns):
00684         """
00685         Virtual method (Order parsing is unnecessary for Icarus, continue to hemisphere
00686         parsing.)
00687         Overridden by Daedalus method
00688         """
00689         return self.ca.partition(frames, columns)
00690
```

6.11.3.8 reportStatusSensor()

```
nsCamera.sensors.sensorBase.sensorBase.reportStatusSensor (
    self,
    statusbits,
    statusbits2 )
```

Print status messages from sensor-specific bits of status register, default for Icarus family sensors

Args:

```
statusbits: result of checkStatus()
statusbits2: result of checkStatus2()
```

Reimplemented in [nsCamera.sensors.daedalus.daedalus](#).

Definition at line 699 of file [sensorBase.py](#).

```
00699     def reportStatusSensor(self, statusbits, statusbits2):
00700         """
00701         Print status messages from sensor-specific bits of status register, default for
00702         Icarus family sensors
00703         Args:
00704             statusbits: result of checkStatus()
00705             statusbits2: result of checkStatus2()
00706         """
00707         if int(statusbits[3]):
00708             print(self.loginfo + "W3_Top_A_Edge1 detected")
00709         if int(statusbits[4]):
00710             print(self.loginfo + "W3_Top_B_Edge1 detected")
00711         if int(statusbits[12]):
00712             print(self.loginfo + "HST_All_W_En detected")
00713         if self.ca.boardname == "llnl_v4" and int(statusbits2[5]):
00714             print(self.loginfo + "PDBIAS Unready")
00715
00716
00717 # TODO: add function to control TIME_ROW_DCD delay
00718
00719 """
00720 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00721 LLNL-CODE-838080
00722
00723 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00724 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00725 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00726 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00727 be made under this license.
00728 """
```

6.11.3.9 selectOscillator()

```
nsCamera.sensors.sensorBase.sensorBase.selectOscillator (
    self,
    osc = None )
```

Selects oscillator to control sensor timing

Overridden in daedalus.py

Args:

osc: 'relaxation'|'ring'|'ringnoosc'|'external', defaults to relaxation

Returns:

error message as string

Reimplemented in [nsCamera.sensors.daedalus.daedalus](#).

Definition at line 651 of file [sensorBase.py](#).

```
00651     def selectOscillator(self, osc=None):
00652         """
00653         Selects oscillator to control sensor timing
00654         Overridden in daedalus.py
00655         Args:
00656             osc: 'relaxation'|'ring'|'ringnoosc'|'external', defaults to relaxation
00657
00658         Returns:
00659             error message as string
00660         """
00661         logging.info(self.loginfo + "selectOscillator; osc = " + str(osc))
00662         if osc is None:
00663             osc = "rel"
00664         osc = str(osc)
00665         if osc.upper()[3] == "REL":
00666             payload = "00"
00667         elif osc.upper()[3] == "RIN":
00668             if "NO" in osc.upper() or "0" in osc:
00669                 payload = "10"
00670             else:
00671                 payload = "01"
00672         elif osc.lower()[3] in ["ext", "phi"]:
00673             payload = "11"
00674         else:
00675             err = (
00676                 self.logerr + "selectOscillator: invalid parameter supplied. "
00677                 "Oscillator selection is unchanged."
00678             )
00679             logging.error(err)
00680             return err
00681         self.ca.setSubregister("OSC_SELECT", payload)
00682
```

6.11.3.10 setArbTiming()

```
nsCamera.sensors.sensorBase.sensorBase.setArbTiming (
    self,
    side = "AB",
    sequence = None )
```

Set arbitrary high-speed timing sequence.

Args:

side: Hemisphere 'A', 'B', 'AB'

sequence: list of arbitrary timing intervals, beginning with initial delay.

The conventional timing (3,2) with delay = 0 would be represented by

[0,3,2,3,2,3,2,3] on icarus devices, [0,3,2,3,2,3] on daedalus. If used

for interlacing or ZDT, you should populate the entire 40-bit register,
e.g., [0,3,2,3,2,3,2,3,2,3,2,3,2,3,2,3,2]

NOTE Icarus sensors generally cannot use 1 ns timing, so should use at
least 2 ns for frames 2 and 3 integration and interframe times (an initial
delay of only 1 ns is acceptable)

NOTE although the Icarus model 1 only images the middle two frames, timing
entries must be provided for all four frames; to implement frame 1 open
for X ns, shutter closed for Y ns, and frame 2 open for Z ns, use the
sequence [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 308 of file [sensorBase.py](#).

```
00308     def setArbTiming(self, side="AB", sequence=None):
00309         """
00310         Set arbitrary high-speed timing sequence.
00311         Args:
00312             side: Hemisphere 'A', 'B', 'AB'
00313             sequence: list of arbitrary timing intervals, beginning with initial delay.
00314                 The conventional timing (3,2) with delay = 0 would be represented by
00315                 [0,3,2,3,2,3] on icarus devices, [0,3,2,3,2,3] on daedalus. If used
00316                 for interlacing or ZDT, you should populate the entire 40-bit register,
00317                 e.g., [0,3,2,3,2,3,2,3,2,3,2,3,2,3,2,3,2]
00318
00319                 *NOTE* Icarus sensors generally cannot use 1 ns timing, so should use at
00320                 least 2 ns for frames 2 and 3 integration and interframe times (an initial
00321                 delay of only 1 ns is acceptable)
00322
00323                 *NOTE* although the Icarus model 1 only images the middle two frames, timing
00324                 entries must be provided for all four frames; to implement frame 1 open
00325                 for X ns, shutter closed for Y ns, and frame 2 open for Z ns, use the
00326                 sequence [0,1,1,X,Y,Z,1,1]
00327
00328                 *WARNING* arbitrary timings will not be restored after a board power cycle
00329
00330         Returns:
00331             list: Actual timing results
00332         """
00333         logging.info(
00334             "{}setArbTiming; side={}, sequence={}".format(self.loginfo, side, sequence)
00335         )
00336         if sequence is None:
00337             if self.sensfam == "Daedalus":
00338                 sequence = [0, 2, 3, 4, 5, 6]
00339             else:
00340                 sequence = [0, 2, 3, 4, 5, 6, 7, 8]
00341         logging.info(
00342             self.loginfo + "HST side " + side.upper() + " (arbitrary): " + str(sequence)
00343         )
00344         if side.upper() == "AB":
00345             err1, _ = self.setArbTiming(side="A", sequence=sequence)
00346             err2, actual = self.setArbTiming(side="B", sequence=sequence)
00347             return err1 + err2, actual
00348         if side.upper() == "A":
00349             lowreg = "HS_TIMING_DATA_ALO"
00350             highreg = "HS_TIMING_DATA_AHI"
00351         elif side.upper() == "B":
00352             lowreg = "HS_TIMING_DATA_BLO"
00353             highreg = "HS_TIMING_DATA_BHI"
00354         else:
00355             err = (
00356                 self.logerr
00357                 + "Invalid sensor side: "
00358                 + side
00359                 + "; timing settings unchanged"
00360             )
00361             logging.error("{}setArbTiming: {}".format(self.logerr, err))
00362             return err, "0000000000"
00363
00364         full140 = [0] * 40
00365         bitlist = []
```

```

00366         flag = 0 # similar to setTiming, but starts with delay
00367
00368         for a in sequence:
00369             add = [flag] * a
00370             bitlist += add
00371             if flag:
00372                 flag = 0
00373             else:
00374                 flag = 1
00375
00376         logging.debug(self.logdebug + "bitlist = " + str(bitlist))
00377         reversedlist = bitlist[39::-1]
00378         full40[-(len(reversedlist) + 1) : -1] = reversedlist
00379         full40bin = "".join(str(x) for x in full40)
00380         logging.debug(self.logdebug + "full40bin = " + str(full40bin))
00381         full40hex = "%x" % int(full40bin, 2)
00382         logging.debug(self.logdebug + "full40hex = " + str(full40hex))
00383         highpart = full40hex[-10:-8].zfill(8)
00384         lowpart = full40hex[-8:].zfill(8)
00385         self.ca.setRegister(lowreg, lowpart)
00386         self.ca.setRegister(highreg, highpart)
00387         # deactivates manual shutter mode if previously engaged
00388         self.ca.setSubregister("MANSHUT_MODE", "0")
00389         self.ca.setSubregister("HST_MODE", "1")
00390         actual = self.getTiming(side, actual=True)
00391         f0delay = sequence[1] + sequence[2]
00392
00393         if self.ca.sensorname == "icarus":
00394             if actual != sequence[:1] + sequence[3:6]:
00395                 logging.warning(
00396                     self.logwarn + "Due to sequence length and use of the Icarus model "
00397                     "1 sensor, the actual timing sequence for side "
00398                     + side
00399                     + " will be "
00400                     + "{"
00401                     + str(actual[0] + f0delay)
00402                     + "}"
00403                     + " "
00404                     + str(actual[1 : 2 * self.nframes])
00405                 )
00406             else:
00407                 logging.warning(
00408                     self.logwarn + "Due to use of the Icarus model 1 sensor, the actual"
00409                     " timing sequence for side "
00410                     + side
00411                     + " will be "
00412                     + "{"
00413                     + str(actual[0] + f0delay)
00414                     + "}"
00415                     + " "
00416                     + str(actual[1 : 2 * self.nframes])
00417                 )
00418         else:
00419             if actual != sequence:
00420                 logging.warning(
00421                     self.logwarn + "Due to sequence length, actual timing sequence "
00422                     "for side "
00423                     + side
00424                     + " will be "
00425                     + "{"
00426                     + str(actual[0])
00427                     + "}"
00428                     + " "
00429                     + str(actual[1 : 2 * self.nframes])
00430                 )
00431         return "", actual
00432

```

6.11.3.11 setExtClk()

```

nsCamera.sensors.sensorBase.sensorBase.setExtClk (
    self,
    delay )

```

Virtual function; feature is not implemented on Icarus
Overridden in daedalus.py

Reimplemented in [nsCamera.sensors.daedalus.daedalus](#).

Definition at line 136 of file [sensorBase.py](#).

```
00136     def setExtClk(self, delay):
00137         """
00138         Virtual function; feature is not implemented on Icarus
00139         Overridden in daedalus.py
00140         """
00141         if delay:
00142             logging.warning(
00143                 self.logwarn + "External Phi Clock is not supported by Icarus sensors. "
00144             )
00145
```

6.11.3.12 setHighFullWell()

```
nsCamera.sensors.sensorBase.sensorBase.setHighFullWell (
    self,
    flag )
```

Virtual function; feature is not implemented on Icarus
Overridden in daedalus.py

Reimplemented in [nsCamera.sensors.daedalus.daedalus](#).

Definition at line 96 of file [sensorBase.py](#).

```
00096     def setHighFullWell(self, flag):
00097         """
00098         Virtual function; feature is not implemented on Icarus
00099         Overridden in daedalus.py
00100         """
00101         if flag:
00102             logging.warning(
00103                 self.logwarn + "HighFullWell mode is not supported by Icarus sensors. "
00104             )
00105
```

6.11.3.13 setInterlacing()

```
nsCamera.sensors.sensorBase.sensorBase.setInterlacing (
    self,
    ifactor )
```

Virtual function; feature is not implemented on Icarus
Overridden in daedalus.py

Returns:
integer 0

Reimplemented in [nsCamera.sensors.daedalus.daedalus](#).

Definition at line 82 of file [sensorBase.py](#).

```
00082     def setInterlacing(self, ifactor):
00083         """
00084         Virtual function; feature is not implemented on Icarus
00085         Overridden in daedalus.py
00086
00087         Returns:
00088             integer 0
00089         """
00090         if ifactor:
00091             logging.warning(
00092                 self.logwarn + "Interlacing is not supported by Icarus sensors. "
00093             )
00094         return 0
00095
```

6.11.3.14 setManualShutters()

```
nsCamera.sensors.sensorBase.sensorBase.setManualShutters (
    self,
    timing = None )
```

Legacy alias for setManualTiming()

Reimplemented in [nsCamera.sensors.daedalus.daedalus](#).

Definition at line 531 of file [sensorBase.py](#).

```
00531     def setManualShutters(self, timing=None):
00532         """
00533         Legacy alias for setManualTiming()
00534         """
00535         self.setManualTiming(timing)
00536
```

6.11.3.15 setManualTiming()

```
nsCamera.sensors.sensorBase.sensorBase.setManualTiming (
    self,
    timing = None )
```

Manual shutter timing, seven intervals to assign to both hemispheres, e.g., [(100,150,100,150,100,150,100)] for frame 0 open for 100 ns, an interframe pause of 50 ns, frame 1 open for 100 ns, etc.
Provide two sets of seven intervals, e.g., [(100,150,100,150,100,150,100), (200,250,200,250,200,250,200)] to program the A and B hemispheres independently

Overridden in daedalus.py

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 the nearest multiple of 25 ns. (Each count = 25 ns. e.g., a request for 140 ns rounds down to a count of '5', which corresponds to 125 ns))

- Minimum timing is 75 ns
- Maximum is $25 * 2^{30}$ ns (approximately 27 seconds)

Args:

timing: 7- or 14-element list (substructure optional) in nanoseconds

Returns:

tuple (error string, response string from final message)

Reimplemented in [nsCamera.sensors.daedalus.daedalus](#).

Definition at line 537 of file [sensorBase.py](#).

```
00537     def setManualTiming(self, timing=None):
00538         """
00539         Manual shutter timing, seven intervals to assign to both hemispheres, e.g.,
00540         [(100,150,100,150,100,150,100)] for frame 0 open for 100 ns, an interframe
00541         pause of 50 ns, frame 1 open for 100 ns, etc.
00542         Provide two sets of seven intervals, e.g., [(100,150,100,150,100,150,100),
00543         (200,250,200,250,200,250,200)] to program the A and B hemispheres
00544         independently
```

```

00545
00546     Overridden in daedalus.py
00547
00548     The timing list is flattened before processing; the suggested tuple structure is
00549     just for clarity (first tuple is A, second is B) and is optional.
00550
00551     The actual timing is rounded down to the nearest multiple of 25 ns. (Each
00552     count = 25 ns. e.g., a request for 140 ns rounds down to a count of '5',
00553     which corresponds to 125 ns)
00554     - Minimum timing is 75 ns
00555     - Maximum is 25 * 2^30 ns (approximately 27 seconds)
00556
00557     Args:
00558         timing: 7- or 14-element list (substructure optional) in nanoseconds
00559
00560     Returns:
00561         tuple (error string, response string from final message)
00562     """
00563     if timing is None:
00564         logging.info(
00565             self.loginfo
00566             + "No manual timing setting provided, defaulting to (100, 150, 100, "
00567             + "150, 100, 150, 100) for both hemispheres"
00568         )
00569         timing = [
00570             (100, 150, 100, 150, 100, 150, 100),
00571             (100, 150, 100, 150, 100, 150, 100),
00572         ]
00573         logging.info(self.loginfo + "Manual shutter sequence: " + str(timing))
00574         flattened = flattenList(timing)
00575         if len(flattened) == 7:
00576             flattened = 2 * flattened
00577         if (
00578             len(flattened) != 14
00579             or not all(isinstance(x, numbers.Real) for x in flattened)
00580             or not all(x >= 75 for x in flattened)
00581             or not all(x <= 26843545600 for x in flattened)
00582         ):
00583             err = self.logerr + "Invalid manual shutter timing list: " + str(timing)
00584             logging.error(err + "; timing settings unchanged")
00585             return err, "00000000"
00586
00587         timecounts = [int(a // 25) for a in flattened]
00588         self.ca.sensmanual = timing
00589         self.ca.sensntiming = {} # clear HST settings from ca object
00590
00591         control_messages = [
00592             ("W0_INTEGRATION", "{0:#0{1}x}".format(timecounts[0], 10)[2:10]),
00593             ("W0_INTERFRAME", "{0:#0{1}x}".format(timecounts[1], 10)[2:10]),
00594             ("W1_INTEGRATION", "{0:#0{1}x}".format(timecounts[2], 10)[2:10]),
00595             ("W1_INTERFRAME", "{0:#0{1}x}".format(timecounts[3], 10)[2:10]),
00596             ("W2_INTEGRATION", "{0:#0{1}x}".format(timecounts[4], 10)[2:10]),
00597             ("W2_INTERFRAME", "{0:#0{1}x}".format(timecounts[5], 10)[2:10]),
00598             ("W3_INTEGRATION", "{0:#0{1}x}".format(timecounts[6], 10)[2:10]),
00599             ("W0_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[7], 10)[2:10]),
00600             ("W0_INTERFRAME_B", "{0:#0{1}x}".format(timecounts[8], 10)[2:10]),
00601             ("W1_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[9], 10)[2:10]),
00602             ("W1_INTERFRAME_B", "{0:#0{1}x}".format(timecounts[10], 10)[2:10]),
00603             ("W2_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[11], 10)[2:10]),
00604             ("W2_INTERFRAME_B", "{0:#0{1}x}".format(timecounts[12], 10)[2:10]),
00605             ("W3_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[13], 10)[2:10]),
00606             ("HST_MODE", "0"),
00607             ("MANSHUT_MODE", "1"),
00608         ]
00609         return self.ca.submitMessages(control_messages, " setManualShutters: ")
00610

```

6.11.3.16 setPhiDelay()

```

nsCamera.sensors.sensorBase.sensorBase.setPhiDelay (
    self,
    delay )

```

Virtual function; feature is not implemented on Icarus
Overridden in daedalus.py

Reimplemented in [nsCamera.sensors.daedalus.daedalus](#).

Definition at line 126 of file [sensorBase.py](#).

```
00126     def setPhiDelay(self, delay):
00127         """
00128         Virtual function; feature is not implemented on Icarus
00129         Overridden in daedalus.py
00130         """
00131         if delay:
00132             logging.warning(
00133                 self.logwarn + "Phi Delay is not supported by Icarus sensors. "
00134             )
00135
```

6.11.3.17 setTiming()

```
nsCamera.sensors.sensorBase.sensorBase.setTiming (
    self,
    side = "AB",
    sequence = None,
    delay = 0 )
```

Sets timing registers based on 'sequence.' Requesting (0,0) timing will clear the timing register.

WARNING if the entire sequence does not fit into the 40-bit register space, then the actual timings generated may differ from those requested. If the timing sequence fits only once into the register space (i.e., for a single frame, open + closed > 20 ns), then the actual timing will be (n, 40-n), irrespective of the 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 for these devices (besides the delay) should be at least 2 ns

Args:

side: Hemisphere 'A', 'B', 'AB'
sequence: two-element tuple of timing durations in ns, e.g., '(5,2)'
delay: initial delay in ns (1 ns delay is acceptable)

Returns:

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

Definition at line 149 of file [sensorBase.py](#).

```
00149     def setTiming(self, side="AB", sequence=None, delay=0):
00150         """
00151         Sets timing registers based on 'sequence.' Requesting (0,0) timing will clear the
00152         timing register.
00153         *WARNING* if the entire sequence does not fit into the 40-bit register space,
00154         then the actual timings generated may differ from those requested. If the
00155         timing sequence fits only once into the register space (i.e., for a single
00156         frame, open + closed > 20 ns), then the actual timing will be (n, 40-n),
00157         irrespective of the setting of second parameter, e.g. (35,1) will actually
00158         result in (35,5) timing.
00159         *NOTE* Icarus sensors generally cannot use 1 ns timing, so all values for these
00160         devices (besides the delay) should be at least 2 ns
00161
00162         Args:
00163             side: Hemisphere 'A', 'B', 'AB'
00164             sequence: two-element tuple of timing durations in ns, e.g., '(5,2)'
00165             delay: initial delay in ns (1 ns delay is acceptable)
00166
00167         Returns:
00168             tuple (error string, 10-character hexadecimal representation of timing
00169                 sequence)
00170         """
00171         logging.info(
```



```

00172         "{}setTiming; side={}, sequence={}, delay={}".format(
00173             self.logininfo, side, sequence, delay
00174         )
00175     )
00176     if sequence is None:
00177         sequence = (3, 2)
00178     if delay is None:
00179         delay = 0
00180     logging.info(
00181         self.logininfo
00182         + "HST side "
00183         + side.upper()
00184         + ": "
00185         + str(sequence)
00186         + "; delay = "
00187         + str(delay)
00188     )
00189     err = ""
00190     if len(sequence) != 2:
00191         err = (
00192             self.logerr
00193             + "Invalid sequence setting for side: "
00194             + side
00195             + "; timing settings are unchanged"
00196         )
00197         logging.error(err)
00198         return err, "0000000000"
00199     if side.upper() == "AB":
00200         err1, _ = self.setTiming(side="A", sequence=sequence, delay=delay)
00201         err2, full40hex = self.setTiming(side="B", sequence=sequence, delay=delay)
00202         return err1 + err2, full40hex
00203     if side.upper() == "A":
00204         lowreg = "HS_TIMING_DATA_ALO"
00205         highreg = "HS_TIMING_DATA_AHI"
00206     elif side.upper() == "B":
00207         lowreg = "HS_TIMING_DATA_BLO"
00208         highreg = "HS_TIMING_DATA_BHI"
00209     else:
00210         err = (
00211             self.logerr
00212             + "setTiming: Invalid sensor side: "
00213             + side
00214             + "; timing settings unchanged"
00215         )
00216         logging.error(err)
00217         return err, "0000000000"
00218     if (sequence[0] + sequence[1]) + delay > 40:
00219         err = (
00220             self.logerr
00221             + "setTiming: Timing sequence is too long to be implemented; "
00222             + "timing settings unchanged "
00223         )
00224         logging.error(err)
00225         return err, "0000000000"
00226
00227     self.ca.senstiming[side.upper()] = (sequence, delay)
00228     self.ca.sensmanual = [] # clear manual settings from ca
00229
00230     full40 = [0] * 40
00231     bitlist = []
00232     flag = 1
00233     sequence = sequence[:2] # TODO: is this redundant?
00234     for a in sequence:
00235         add = [flag] * a
00236         bitlist += add
00237         if flag:
00238             flag = 0
00239         else:
00240             flag = 1
00241     # automatically truncates sequence to 39 characters
00242     logging.debug(self.logdebug + "bitlist = " + str(bitlist))
00243     if bitlist: # skip this if timing is [0,0]
00244         reversedlist = bitlist[39::-1]
00245         trunclist = reversedlist[:]
00246         while trunclist[0] == 0:
00247             trunclist.pop(0)
00248         # fullrepeat counts open/closed cycles, doesn't include final frame
00249         fullrepeats = (40 - len(trunclist) - delay) // len(reversedlist)
00250         logging.debug(self.logdebug + "fullrepeats = " + str(fullrepeats))
00251         # Pattern from sequence repeated to fit inside 40 bits
00252         repeated = trunclist + reversedlist * fullrepeats

```

```

00253         full40[-(len(repeated) + delay + 1) : -(delay + 1)] = repeated
00254     else:
00255         logging.warning(self.logwarn + "setTiming: all-zero timing supplied")
00256         fullrepeats = self.nframes
00257         full40bin = "".join(str(x) for x in full40)
00258         logging.debug(self.logdebug + "full40bin = " + str(full40bin))
00259         full40hex = "%x" % int(full40bin, 2)
00260         logging.debug(self.logdebug + "full40hex = " + str(full40hex))
00261         highpart = full40hex[-10:-8].zfill(8)
00262         lowpart = full40hex[-8:].zfill(8)
00263         err0, _ = self.ca.setRegister(lowreg, lowpart)
00264         err1, _ = self.ca.setRegister(highreg, highpart)
00265         err2, _ = self.ca.setSubregister("MANSHUT_MODE", "0")
00266         err3, _ = self.ca.setSubregister("HST_MODE", "1")
00267         err = err0 + err1 + err2 + err3
00268     if err:
00269         logging.error(
00270             self.logerr + "setTiming: Timing may not have been set correctly"
00271         )
00272     if fullrepeats < self.nframes - 1:
00273         actual = self.getTiming(side, actual=True)
00274         if self.fpganumID == 2:
00275             expected = [delay] + 2 * list(sequence) + [sequence[0]]
00276         else:
00277             expected = [delay] + 3 * list(sequence) + [sequence[0]]
00278         if actual != expected:
00279             logging.warning(
00280                 self.logwarn
00281                 + "setTiming: Due to sequence length "
00282                 + self.specwarn
00283                 + ", the actual timing "
00284                 + "sequence for side "
00285                 + side
00286                 + " will be "
00287                 + "{"
00288                 + str(actual[0])
00289                 + "}"
00290                 + " "
00291                 + str(actual[1 : 2 * self.nframes])
00292             )
00293     elif self.ca.sensorname == "icarus":
00294         f0delay = sequence[0] + sequence[1]
00295         logging.warning(
00296             self.logwarn + "setTiming: Due to use of the Icarus model 1 sensor, the"
00297             + " initial delay for side "
00298             + side
00299             + " will actually be "
00300             + str(delay + f0delay)
00301             + " nanoseconds"
00302         )
00303     return err, full40hex
00304

```

6.11.3.18 setTriggerDelay()

```

nsCamera.sensors.sensorBase.sensorBase.setTriggerDelay (
    self,
    delay )

```

Virtual function; feature is not implemented on Icarus
Overridden in daedalus.py

Reimplemented in [nsCamera.sensors.daedalus.daedalus](#).

Definition at line 116 of file [sensorBase.py](#).

```

00116     def setTriggerDelay(self, delay):
00117         """
00118         Virtual function; feature is not implemented on Icarus
00119         Overridden in daedalus.py
00120         """
00121         if delay:
00122             logging.warning(
00123                 self.logwarn + "Trigger Delay is not supported by Icarus sensors. "
00124             )
00125

```

6.11.3.19 setZeroDeadTime()

```
nsCamera.sensors.sensorBase.sensorBase.setZeroDeadTime (
    self,
    flag )
```

Virtual function; feature is not implemented on Icarus
Overridden in daedalus.py

Reimplemented in [nsCamera.sensors.daedalus.daedalus](#).

Definition at line 106 of file [sensorBase.py](#).

```
00106     def setZeroDeadTime(self, flag):
00107         """
00108         Virtual function; feature is not implemented on Icarus
00109         Overridden in daedalus.py
00110         """
00111         if flag:
00112             logging.warning(
00113                 self.logwarn + "ZeroDeadTime mode is not supported by Icarus sensors. "
00114             )
00115
```

6.11.4 Member Data Documentation

6.11.4.1 ca

```
nsCamera.sensors.sensorBase.sensorBase.ca
```

Definition at line 32 of file [sensorBase.py](#).

6.11.4.2 fpganumID

```
nsCamera.sensors.sensorBase.sensorBase.fpganumID
```

Definition at line 274 of file [sensorBase.py](#).

6.11.4.3 logcrit

```
nsCamera.sensors.sensorBase.sensorBase.logcrit
```

Definition at line 38 of file [sensorBase.py](#).

6.11.4.4 logdebug

```
nsCamera.sensors.sensorBase.sensorBase.logdebug
```

Definition at line 42 of file [sensorBase.py](#).

6.11.4.5 logerr

`nsCamera.sensors.sensorBase.sensorBase.logerr`

Definition at line 39 of file [sensorBase.py](#).

6.11.4.6 loginfo

`nsCamera.sensors.sensorBase.sensorBase.loginfo`

Definition at line 41 of file [sensorBase.py](#).

6.11.4.7 logwarn

`nsCamera.sensors.sensorBase.sensorBase.logwarn`

Definition at line 40 of file [sensorBase.py](#).

6.11.4.8 sensfam

`nsCamera.sensors.sensorBase.sensorBase.sensfam`

Definition at line 337 of file [sensorBase.py](#).

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

- C:/Users/hill35/git/camera_python/nsCamera/sensors/[sensorBase.py](#)

6.12 nsCamera.utils.Subregister.SubRegister Class Reference

Public Member Functions

- `__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](#)

6.12.1 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](#).

6.12.2 Constructor & Destructor Documentation

6.12.2.1 __init__()

```
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](#).

```
00039     ):
00040         self.name = name
00041         self.register = register
00042         self.addr = board.registers[register]
00043         self.start_bit = start_bit
00044         self.width = width
00045         self.value = value
00046         self.max_value = 2**width - 1 # used to normalize the input values to 1
00047         self.min = 0
00048         self.max = self.max_value
00049         self.writable = writable
00050         self.minV = minV
00051         self.maxV = maxV
00052         # resolution should be reset after init if actual min and max are different
00053         self.resolution = (1.0 * maxV - minV) / self.max_value
00054
00055     """
00056
00057     Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00058     LLNL-CODE-838080
00059
00060     This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00061     contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00062     and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00063     'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00064     be made under this license.
00065     """
```

6.12.3 Member Data Documentation

6.12.3.1 addr

```
nsCamera.utils.Subregister.SubRegister.addr
```

Definition at line 42 of file [Subregister.py](#).

6.12.3.2 max

`nsCamera.utils.Subregister.SubRegister.max`

Definition at line 48 of file [Subregister.py](#).

6.12.3.3 max_value

`nsCamera.utils.Subregister.SubRegister.max_value`

Definition at line 46 of file [Subregister.py](#).

6.12.3.4 maxV

`nsCamera.utils.Subregister.SubRegister.maxV`

Definition at line 51 of file [Subregister.py](#).

6.12.3.5 min

`nsCamera.utils.Subregister.SubRegister.min`

Definition at line 47 of file [Subregister.py](#).

6.12.3.6 minV

`nsCamera.utils.Subregister.SubRegister.minV`

Definition at line 50 of file [Subregister.py](#).

6.12.3.7 name

`nsCamera.utils.Subregister.SubRegister.name`

Definition at line 40 of file [Subregister.py](#).

6.12.3.8 register

`nsCamera.utils.Subregister.SubRegister.register`

Definition at line 41 of file [Subregister.py](#).

6.12.3.9 resolution

`nsCamera.utils.Subregister.SubRegister.resolution`

Definition at line 53 of file [Subregister.py](#).

6.12.3.10 start_bit

`nsCamera.utils.Subregister.SubRegister.start_bit`

Definition at line 43 of file [Subregister.py](#).

6.12.3.11 value

`nsCamera.utils.Subregister.SubRegister.value`

Definition at line 45 of file [Subregister.py](#).

6.12.3.12 width

`nsCamera.utils.Subregister.SubRegister.width`

Definition at line 44 of file [Subregister.py](#).

6.12.3.13 writable

`nsCamera.utils.Subregister.SubRegister.writable`

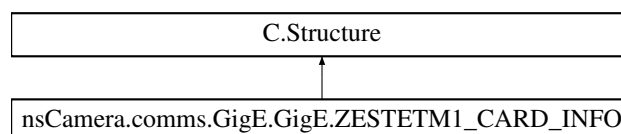
Definition at line 49 of file [Subregister.py](#).

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

- [C:/Users/hill35/git/camera_python/nsCamera/utils/Subregister.py](#)

6.13 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 Protected Attributes

- list [_fields_](#)

6.13.1 Detailed Description

Definition at line 512 of file [GigE.py](#).

6.13.2 Member Data Documentation

6.13.2.1 [_fields_](#)

```
list nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO._fields_ [static], [protected]
```

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 515 of file [GigE.py](#).

6.13.2.2 [ubyte4](#)

```
int nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO.ubyte4 = C.c_ubyte * 4 [static]
```

Definition at line 513 of file [GigE.py](#).

6.13.2.3 [ubyte6](#)

```
int nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO.ubyte6 = C.c_ubyte * 6 [static]
```

Definition at line 514 of file [GigE.py](#).

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

- C:/Users/hill35/git/camera_python/nsCamera/comms/[GigE.py](#)

Chapter 7

File Documentation

7.1 C:/Users/hill35/git/camera_python/nsCamera/__init__.py File Reference

Namespaces

- namespace `nsCamera`

Variables

- list `nsCamera.__all__` = ["CameraAssembler"]

7.2 __init__.py

[Go to the documentation of this file.](#)

```
00001 # -*- coding: utf-8 -*-
00002 """
00003 Created on Tue Mar 22 15:47:43 2016
00004
00005 The Package includes a Camera object and an assembler.
00006
00007 The camera object will be the workhorse of the API. The assembler is used
00008 to create the Camera object.
00009
00010 Author: Matthew Dayton (dayton5@llnl.gov)
00011
00012 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00013 LLNL-CODE-838080
00014
00015 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00016 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00017 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00018 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00019 be made under this license.
00020
00021 Version: 2.1.2 (February 2025)
00022 """
00023
00024 from nsCamera import CameraAssembler
00025
00026 __all__ = ["CameraAssembler"]
00027
00028 """
```

```

00029 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00030 LLNL-CODE-838080
00031
00032 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00033 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00034 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00035 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00036 be made under this license.
00037 """

```

7.3 C:/Users/hill35/git/camera_python/nsCamera/boards/__init__.py File Reference

Namespaces

- namespace `nsCamera`
- namespace `nsCamera.boards`

Variables

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

7.4 __init__.py

[Go to the documentation of this file.](#)

```

00001 # -*- coding: utf-8 -*-
00002 """
00003 This package is a collection of modules that represent the camera boards
00004 Each board has its own number of ADCs, POTs, and sensors. More devices can be added in
00005 the future. The list of imports will grow as we make more types of boards.
00006
00007 Author: Matthew Dayton (dayton5@llnl.gov)
00008
00009 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00010 LLNL-CODE-838080
00011
00012 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00013 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00014 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00015 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00016 be made under this license.
00017
00018 Version: 2.1.2 (February 2025)
00019 """
00020 from .LLNL_v1 import llnl_v1
00021 from .LLNL_v4 import llnl_v4
00022
00023 __all__ = ["LLNL_v1", "LLNL_v4"]
00024
00025 """
00026 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00027 LLNL-CODE-838080
00028
00029 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00030 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00031 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00032 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00033 be made under this license.
00034 """

```

7.5 C:/Users/hill35/git/camera_python/nsCamera/comms/__init__.py File Reference

Namespaces

- namespace [nsCamera](#)
- namespace [nsCamera.comms](#)

Variables

- list [nsCamera.comms.__all__](#) = ["RS422", "GigE"]

7.6 __init__.py

[Go to the documentation of this file.](#)

```
00001 # -*- coding: utf-8 -*-
00002 """
00003 Created on Tue Mar 22 15:47:43 2016
00004
00005 This package is a collection of modules for uniform handling of the nsCamera's
00006 communication systems
00007
00008 Author: Matthew Dayton (dayton5@llnl.gov)
00009
00010 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00011 LLNL-CODE-838080
00012
00013 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00014 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00015 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00016 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00017 be made under this license.
00018
00019 Version: 2.1.2 (February 2025)
00020 """
00021
00022 from .GigE import GigE
00023 from .RS422 import RS422
00024
00025 __all__ = ["RS422", "GigE"]
00026
00027 """
00028 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00029 LLNL-CODE-838080
00030
00031 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00032 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00033 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00034 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00035 be made under this license.
00036 """
```

7.7 C:/Users/hill35/git/camera_python/nsCamera/sensors/__init__.py File Reference

Namespaces

- namespace [nsCamera](#)
- namespace [nsCamera.sensors](#)

Variables

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

7.8 `__init__.py`

[Go to the documentation of this file.](#)

```
00001 # -*- coding: utf-8 -*-
00002 """
00003 This package is a collection of modules for uniform handling of the various sensors
00004
00005 Author: Jeremy Martin Hill (jerhill@llnl.gov)
00006
00007 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00008 LLNL-CODE-838080
00009
00010 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00011 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00012 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00013 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00014 be made under this license.
00015
00016 Version: 2.1.2 (February 2025)
00017 """
00018
00019 from .daedalus import daedalus
00020 from .icarus import icarus
00021 from .icarus2 import icarus2
00022
00023 __all__ = ["icarus", "icarus2", "daedalus"]
00024
00025 """
00026 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00027 LLNL-CODE-838080
00028
00029 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00030 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00031 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00032 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00033 be made under this license.
00034 """
```

7.9 `C:/Users/hill35/git/camera_python/nsCamera/utils/__init__.py` File Reference

Namespaces

- namespace `nsCamera`
- namespace `nsCamera.utils`

Variables

- list `nsCamera.utils.__all__` = ["SubRegister", "Packet", "FlatField", "misc"]

7.10 __init__.py

[Go to the documentation of this file.](#)

```
00001 # -*- coding: utf-8 -*-
00002 """
00003 This package is a collection of utility classes for the CameraAssembler
00004
00005 Author: Jeremy Martin Hill (jerhill@llnl.gov)
00006
00007 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00008 LLNL-CODE-838080
00009
00010 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00011 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00012 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00013 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00014 be made under this license.
00015
00016 Version: 2.1.2 (February 2025)
00017 """
00018
00019 from .Packet import Packet
00020 from .Subregister import SubRegister
00021
00022 try:
00023     from misc import *
00024     from .crcl6pure import *
00025     from .FlatField import *
00026
00027 except:
00028     pass
00029
00030 __all__ = ["SubRegister", "Packet", "FlatField", "misc"]
00031
00032 """
00033 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00034 LLNL-CODE-838080
00035
00036 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00037 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00038 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00039 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00040 be made under this license.
00041 """
```

7.11 C:/Users/hill35/git/camera_python/nsCamera/boards/LLNL_v1.py File Reference

Classes

- class [nsCamera.boards.LLNL_v1.lnl_v1](#)

Namespaces

- namespace [nsCamera](#)
- namespace [nsCamera.boards](#)
- namespace [nsCamera.boards.LLNL_v1](#)

7.12 LLNL_v1.py

[Go to the documentation of this file.](#)

```
00001 # -*- coding: utf-8 -*-
00002 """
00003 LLNLv1 board definition, including monitors, pots, and other board-specific settings
00004
00005 Author: Jeremy Martin Hill (jerhill@llnl.gov)
00006 Author: Matthew Dayton (dayton5@llnl.gov)
00007
00008 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00009 LLNL-CODE-838080
00010
00011 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00012 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00013 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00014 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00015 be made under this license.
00016
00017 Version: 2.1.2 (February 2025)
00018 """
00019
00020 import logging
00021 import time
00022 from collections import OrderedDict
00023
00024 from nsCamera.utils.Packet import Packet
00025 from nsCamera.utils.Subregister import SubRegister
00026
00027
00028 class llnl_v1:
00029     """
00030     Livermore LLNL v1.0 board
00031
00032     Compatible communication protocols: RS422, GigE
00033     Compatible sensors: icarus, icarus2, daedalus
00034     """
00035
00036     # FPGA register map - use '.upper()' on keys to ensure uppercase lookup
00037     registers = OrderedDict(
00038         {
00039             "FPGA_NUM": "000",
00040             "FPGA_REV": "001",
00041             "HS_TIMING_CTL": "010",
00042             "HS_TIMING_DATA_ALO": "013",
00043             "HS_TIMING_DATA_AHI": "014",
00044             "HS_TIMING_DATA_BLO": "015",
00045             "HS_TIMING_DATA_BHI": "016",
00046             "SW_TRIGGER_CONTROL": "017",
00047             "STAT_REG": "024",
00048             "CTRL_REG": "025",
00049             "POT_CTL": "026",
00050             "POT_REG4_TO_1": "027",
00051             "POT_REG8_TO_5": "028",
00052             "POT_REG12_TO_9": "029",
00053             "POT_REG13": "02A",
00054             "LED_GP": "02B",
00055             "SW_RESET": "02D",
00056             "HST_SETTINGS": "02E",
00057             "STAT_REG_SRC": "02F",
00058             "STAT_REG2": "030",
00059             "STAT_REG2_SRC": "031",
00060             "ADC_BYTECOUNTER": "032",
00061             "RBP_PIXEL_CNTR": "033",
00062             "DIAG_MAX_CNT_0": "034",
00063             "DIAG_MAX_CNT_1": "035",
00064             "DIAG_CNTR_VAL_0": "036",
00065             "DIAG_CNTR_VAL_1": "037",
00066             "TRIGGER_CTL": "03A",
00067             "SRAM_CTL": "03B",
00068             "TIMER_CTL": "03C",
00069             "TIMER_VALUE": "03D",
00070             "HSTALLWEN_WAIT_TIME": "03F",
00071             "FPA_ROW_INITIAL": "042",
00072             "FPA_ROW_FINAL": "043",
00073             "FPA_FRAME_INITIAL": "044",
00074             "FPA_FRAME_FINAL": "045",
00075             "FPA_DIVCLK_EN_ADDR": "046",
00076             "FPA_OSCILLATOR_SEL_ADDR": "047",
```

```

00077         "FRAME_ORDER_SEL": "04B",
00078         "SENSOR_VOLT_STAT": "082",
00079         "SENSOR_VOLT_CTL": "083",
00080         "ADC_CTL": "090",
00081         "ADC1_CONFIG_DATA": "091",
00082         "ADC2_CONFIG_DATA": "092",
00083         "ADC3_CONFIG_DATA": "093",
00084         "ADC4_CONFIG_DATA": "094",
00085         "ADC5_CONFIG_DATA": "095",
00086         "ADC5_DATA_1": "096",
00087         "ADC5_DATA_2": "097",
00088         "ADC5_DATA_3": "098",
00089         "ADC5_DATA_4": "099",
00090         "ADC5_PPER": "09A",
00091         "ADC_STANDBY": "09B", # board version <= AD
00092         "ADC_RESET": "09B", # board version > AD
00093         "TEMP_SENSE_PPER": "0A0",
00094         "TEMP_SENSE_DATA": "0A1",
00095     }
00096 )
00097
00098 subregisters = [
00099
00100     ("HST_MODE", "HS_TIMING_CTL", 0, 1, True),
00101     ("SW_TRIG_START", "SW_TRIGGER_CONTROL", 0, 1, True),
00102     ("LED_EN", "CTRL_REG", 1, 1, True),
00103     ("COLQUENCHEN", "CTRL_REG", 2, 1, True),
00104     ("POWERSAVE", "CTRL_REG", 3, 1, True),
00105     ("POT1", "POT_REG4_TO_1", 7, 8, True),
00106     ("POT2", "POT_REG4_TO_1", 15, 8, True),
00107     ("POT3", "POT_REG4_TO_1", 23, 8, True),
00108     ("POT4", "POT_REG4_TO_1", 31, 8, True),
00109     ("POT5", "POT_REG8_TO_5", 7, 8, True),
00110     ("POT6", "POT_REG8_TO_5", 15, 8, True),
00111     ("POT7", "POT_REG8_TO_5", 23, 8, True),
00112     ("POT8", "POT_REG8_TO_5", 31, 8, True),
00113     ("POT9", "POT_REG12_TO_9", 7, 8, True),
00114     ("POT10", "POT_REG12_TO_9", 15, 8, True),
00115     ("POT11", "POT_REG12_TO_9", 23, 8, True),
00116     ("POT12", "POT_REG12_TO_9", 31, 8, True),
00117     ("POT13", "POT_REG13", 7, 8, True),
00118     ("LED1", "LED_GP", 0, 1, True),
00119     ("LED2", "LED_GP", 1, 1, True),
00120     ("LED3", "LED_GP", 2, 1, True),
00121     ("LED4", "LED_GP", 3, 1, True),
00122     ("LED5", "LED_GP", 4, 1, True),
00123     ("LED6", "LED_GP", 5, 1, True),
00124     ("LED7", "LED_GP", 6, 1, True),
00125     ("LED8", "LED_GP", 7, 1, True),
00126     ("RESET", "SW_RESET", 0, 1, True),
00127     ("HST_SW_CTL_EN", "HST_SETTINGS", 0, 1, True),
00128     ("SW_HSTALLWEN", "HST_SETTINGS", 1, 1, True),
00129     ("MAXERR_FIT", "DIAG_MAX_CNT_0", 31, 16, True),
00130     ("MAXERR_SRT", "DIAG_MAX_CNT_0", 7, 8, True),
00131     ("MAXERR_UTTR", "DIAG_MAX_CNT_1", 31, 16, True),
00132     ("MAXERR_URTR", "DIAG_MAX_CNT_1", 15, 16, True),
00133     ("HW_TRIG_EN", "TRIGGER_CTL", 0, 1, True),
00134     ("SW_TRIG_EN", "TRIGGER_CTL", 2, 1, True),
00135     ("READ_SRAM", "SRAM_CTL", 0, 1, True),
00136     ("RESET_TIMER", "TIMER_CTL", 0, 1, True),
00137     ("OSC_SELECT", "FPA_OSCILLATOR_SEL_ADDR", 1, 2, True),
00138     ("ADC5_VREF", "ADC5_CONFIG_DATA", 9, 10, True),
00139     ("ADC5_VREF3", "ADC5_CONFIG_DATA", 13, 1, True),
00140     ("ADC5_INT", "ADC5_CONFIG_DATA", 15, 1, True),
00141     ("ADC5_MULT", "ADC5_CONFIG_DATA", 24, 6, True),
00142     ("PPER", "ADC5_PPER", 7, 8, True),
00143
00144     ("SRAM_READY", "STAT_REG", 0, 1, False),
00145     ("STAT_COARSE", "STAT_REG", 1, 1, False),
00146     ("STAT_FINE", "STAT_REG", 2, 1, False),
00147     ("STAT_SENSREADIP", "STAT_REG", 5, 1, False),
00148     ("STAT_SENSREADDONE", "STAT_REG", 6, 1, False),
00149     ("STAT_SRAMREADSTART", "STAT_REG", 7, 1, False),
00150     ("STAT_SRAMREADDONE", "STAT_REG", 8, 1, False),
00151     ("STAT_HSTCONFIGSTART", "STAT_REG", 9, 1, False),
00152     ("STAT_ADCSCONFIGURED", "STAT_REG", 10, 1, False),
00153     ("STAT_POTSCONFIGURED", "STAT_REG", 11, 1, False),
00154     ("STAT_TIMERCOUNTERRESET", "STAT_REG", 13, 1, False),
00155     ("STAT_ARMED", "STAT_REG", 14, 1, False),
00156     ("STAT_TEMP", "STAT_REG", 27, 11, False),
00157     ("STAT_PRESS", "STAT_REG", 31, 4, False),
00158 ]

```

```

00165         ("FPA_IF_TO", "STAT_REG2", 0, 1, False),
00166         ("SRAM_RO_TO", "STAT_REG2", 1, 1, False),
00167         ("PIXELRD_TOUT_ERR", "STAT_REG2", 2, 1, False),
00168         ("UART_TX_TO_RST", "STAT_REG2", 3, 1, False),
00169         ("UART_RX_TO_RST", "STAT_REG2", 4, 1, False),
00170         ("SENSOR_POSN", "SENSOR_VOLT_STAT", 0, 1, False),
00171         ("SENSOR_NEGP", "SENSOR_VOLT_STAT", 1, 1, False),
00172         ("ICARUS_DET", "SENSOR_VOLT_STAT", 2, 1, False),
00173         ("DAEDALUS_DET", "SENSOR_VOLT_STAT", 3, 1, False),
00174         ("HORUS_DET", "SENSOR_VOLT_STAT", 4, 1, False),
00175         ("SENSOR_POWER", "SENSOR_VOLT_STAT", 5, 1, False),
00176         ("FIT_COUNT", "DIAG_CNTR_VAL_0", 31, 16, False),
00177         ("SRT_COUNT", "DIAG_CNTR_VAL_0", 7, 8, False),
00178         ("UTTR_COUNT", "DIAG_CNTR_VAL_1", 31, 16, False),
00179         ("URTR_COUNT", "DIAG_CNTR_VAL_1", 15, 16, False),
00180         # monitor ADC channels defined here - the poll period will need to be set during
00181         # camera initialization (x98)
00182         ("MON_CH2", "ADC5_DATA_1", 15, 16, False),
00183         ("MON_CH3", "ADC5_DATA_1", 31, 16, False),
00184         ("MON_CH4", "ADC5_DATA_2", 15, 16, False),
00185         ("MON_CH5", "ADC5_DATA_2", 31, 16, False),
00186         ("MON_CH6", "ADC5_DATA_3", 15, 16, False),
00187         ("MON_CH7", "ADC5_DATA_3", 31, 16, False),
00188         ("MON_CH8", "ADC5_DATA_4", 15, 16, False),
00189         ("MON_VRST", "ADC5_DATA_4", 31, 16, False),
00190     ]
00191
00192     def __init__(self, camassem):
00193         self.ca = camassem
00194         self.logcrit = self.ca.logcritbase + "[LLNL_v1] "
00195         self.logerr = self.ca.logerrbase + "[LLNL_v1] "
00196         self.logwarn = self.ca.logwarnbase + "[LLNL_v1] "
00197         self.loginfo = self.ca.loginfobase + "[LLNL_v1] "
00198         self.logdebug = self.ca.logdebugbase + "[LLNL_v1] "
00199         logging.info(self.loginfo + "initializing board object")
00200         self.VREF = 2.5 # default
00201         self.ADC5_mult = 2 # i.e., monmax = 2 * VREF
00202         # False => monitor range runs 0 to monmax, True => +/- monmax
00203         self.ADC5_bipolar = True
00204         self.rs422_baud = 921600
00205         self.rs422_cmd_wait = 0.3
00206
00207         fpgaNum_pkt = Packet(cmd="1", addr=self.registers["FPGA_NUM"])
00208         fpgaRev_pkt = Packet(cmd="1", addr=self.registers["FPGA_REV"])
00209
00210         _, _ = self.ca.sendCMD(fpgaNum_pkt) # dummy duplicate call
00211         err, rval = self.ca.sendCMD(fpgaNum_pkt)
00212         self.ca.FPGA_NUM = rval[8:16]
00213
00214         err, rval = self.ca.sendCMD(fpgaRev_pkt)
00215         self.ca.FPGA_VERSION = rval[8:16]
00216
00217         # map channels to signal names for abstraction at the camera assembler level;
00218         # each requires a corresponding entry in 'subregisters'
00219
00220         self.icarus_subreg_aliases = OrderedDict(
00221             {
00222                 "COL_BOT_IBIAS_IN": "POT1",
00223                 "HST_A_PDELAY": "POT2",
00224                 "HST_B_NDELAY": "POT3",
00225                 "HST_RO_IBIAS": "POT4",
00226                 "HST_OSC_VREF_IN": "POT5",
00227                 "HST_B_PDELAY": "POT6",
00228                 "HST_OSC_CTL": "POT7",
00229                 "HST_A_NDELAY": "POT8",
00230                 "COL_TOP_IBIAS_IN": "POT9",
00231                 "HST_OSC_R_BIAS": "POT10",
00232                 "VAB": "POT11",
00233                 "HST_RO_NC_IBIAS": "POT12",
00234                 "VRST": "POT13",
00235                 "MON_HST_A_PDELAY": "MON_CH2",
00236                 "MON_HST_B_NDELAY": "MON_CH3",
00237                 "MON_HST_RO_IBIAS": "MON_CH4",
00238                 "MON_HST_OSC_VREF_IN": "MON_CH5",
00239                 "MON_HST_B_PDELAY": "MON_CH6",
00240                 "MON_HST_OSC_CTL": "MON_CH7",
00241                 "MON_HST_A_NDELAY": "MON_CH8",
00242             }
00243         )
00244         # Read-only; identifies controls corresponding to monitors
00245         self.icarus_monitor_controls = OrderedDict(

```



```

00246         {
00247             "MON_CH2": "POT2",
00248             "MON_CH3": "POT3",
00249             "MON_CH4": "POT4",
00250             "MON_CH5": "POT5",
00251             "MON_CH6": "POT6",
00252             "MON_CH7": "POT7",
00253             "MON_CH8": "POT8",
00254             # Note: VRST is not measured across the pot; it will read a voltage
00255             # approximately 1 Volt lower than pot13's actual output
00256             "MON_VRST": "POT13",
00257         }
00258     )
00259
00260     self.daedalus_subreg_aliases = OrderedDict(
00261         {
00262             "HST_OSC_CTL": "POT4",
00263             "HST_RO_NC_IBIAS": "POT5",
00264             "HST_OSC_VREF_IN": "POT6",
00265             "VAB": "POT11",
00266             "MON_TSENSEOUT": "MON_CH2",
00267             "MON_BGREF": "MON_CH3",
00268             "MON_HST_OSC_CTL": "MON_CH4",
00269             "MON_HST_RO_NC_IBIAS": "MON_CH5",
00270             "MON_HST_OSC_VREF_IN": "MON_CH6",
00271             "MON_COL_TST_IN": "MON_CH7",
00272             "MON_HST_OSC_PBIAS_PAD": "MON_CH8",
00273         }
00274     )
00275     # Read-only; identifies controls corresponding to monitors
00276     self.daedalus_monitor_controls = OrderedDict(
00277         {
00278             "MON_CH4": "POT4",
00279             "MON_CH5": "POT5",
00280             "MON_CH6": "POT6",
00281             # Note: VRST is not measured across the pot; it will read a voltage
00282             # lower than pot13's actual output
00283             "MON_VRST": "POT13",
00284         }
00285     )
00286
00287     self.subreglist = []
00288     for s in self.subregisters:
00289         self.subreglist.append(s[0].upper())
00290         sr = SubRegister(
00291             self,
00292             name=s[0].upper(),
00293             register=s[1].upper(),
00294             start_bit=s[2],
00295             width=s[3],
00296             writable=s[4],
00297         )
00298         setattr(self, s[0].upper(), sr)
00299
00300     # set voltage ranges for all pots
00301     for n in range(1, 13):
00302         potname = "POT" + str(n)
00303         potobj = getattr(self, potname)
00304         potobj.minV = 0
00305         potobj.maxV = 3.3
00306         # resolution is approximately .0129 V / LSB
00307         potobj.resolution = (1.0 * potobj.maxV - potobj.minV) / potobj.max_value
00308     self.POT13.minV = 0
00309     self.POT13.maxV = 3.96
00310     # POT13 resolution is approximately .0155 V / LSB
00311     self.POT13.resolution = (
00312         1.0 * self.POT13.maxV - self.POT13.minV
00313     ) / self.POT13.max_value
00314
00315     def initBoard(self):
00316         """
00317         Register and reset board, set up firmware for sensor
00318
00319         Returns:
00320             tuple (error string, response string) from final control message
00321         """
00322         logging.info(self.loginfo + "initBoard LLNLv1")
00323         control_messages = [{"LED_EN", "1"}]
00324
00325         self.clearStatus()
00326         self.configADCs()

```

```

00327
00328     err, resp = self.ca.getSubregister("ADC5_VREF3")
00329     if err:
00330         logging.error(self.logerr + "unable to read 'ADC5_VREF3'")
00331     if int(resp, 2): # check to see if Vref is 3 or 2.5 volts
00332         vrefmax = 3.0
00333     else:
00334         vrefmax = 2.5
00335     err, resp = self.ca.getSubregister("ADC5_VREF")
00336     if err:
00337         logging.error(self.logerr + "unable to read 'ADC5_VREF'")
00338     self.VREF = vrefmax * int(resp, 2) / 1024.0
00339     err, multmask = self.ca.getSubregister("ADC5_MULT")
00340     if err:
00341         logging.error(self.logerr + "unable to read 'ADC5_MULT'")
00342     if multmask[0] and multmask[1] and multmask[3] and multmask[5]:
00343         self.ADC5_mult = 2
00344     elif not (multmask[0] or multmask[1] or multmask[3] or multmask[5]):
00345         self.ADC5_mult = 4
00346     else:
00347         logging.error(self.logerr + "inconsistent mode settings on ADC5")
00348     return self.ca.submitMessages(control_messages, " initBoard: ")
00349
00350 def initPots(self):
00351     """
00352     Configure default pot settings before image acquisition
00353
00354     Returns:
00355         tuple (error string, response string) from final control message
00356     """
00357     logging.info(self.loginf + "initPots")
00358     if self.ca.sensorname == "icarus" or self.ca.sensorname == "icarus2":
00359         err0, _ = self.ca.setPot("HST_A_PDELAY", 0, errflag=True)
00360         err1, _ = self.ca.setPotV("HST_B_NDELAY", 3.3, errflag=True)
00361         err2, _ = self.ca.setPotV("HST_RO_IBIAS", 2.5, tune=True, errflag=True)
00362         err3, _ = self.ca.setPotV("HST_OSC_VREF_IN", 2.9, tune=True, errflag=True)
00363         err4, _ = self.ca.setPot("HST_B_PDELAY", 0, errflag=True)
00364         err5, _ = self.ca.setPotV("HST_OSC_CTL", 1.45, tune=True, errflag=True)
00365         err6, _ = self.ca.setPotV("HST_A_NDELAY", 3.3, errflag=True)
00366         err7, _ = self.ca.setPotV("VAB", 0.5, errflag=True)
00367         err8, _ = self.ca.setPotV("HST_RO_NC_IBIAS", 2.5, errflag=True)
00368         err9, _ = self.ca.setPotV("VRST", 0.3, tune=True, errflag=True)
00369         err = err0 + err1 + err2 + err3 + err4 + err5 + err6 + err7 + err8 + err9
00370     else: # Daedalus
00371         err0, _ = self.ca.setPotV("HST_OSC_CTL", 1.0, tune=True, errflag=True)
00372         err1, _ = self.ca.setPotV("HST_RO_NC_IBIAS", 1.0, errflag=True)
00373         err2, _ = self.ca.setPotV("HST_OSC_VREF_IN", 1.0, tune=True, errflag=True)
00374         err3, _ = self.ca.setPotV("VAB", 0.5, errflag=True)
00375         err = err0 + err1 + err2 + err3
00376     return err, ""
00377
00378 def latchPots(self):
00379     """
00380     Latch pot settings into sensor
00381
00382     Returns:
00383         tuple (error string, response string) from final control message
00384     """
00385     logging.info(self.loginf + "latchPots")
00386
00387     control_messages = [
00388         ("POT_CTL", "00000003"), # latches register settings for pot 1
00389         ("POT_CTL", "00000005"),
00390         ("POT_CTL", "00000007"),
00391         ("POT_CTL", "00000009"),
00392         ("POT_CTL", "0000000B"),
00393         ("POT_CTL", "0000000D"),
00394         ("POT_CTL", "0000000F"),
00395         ("POT_CTL", "00000011"),
00396         ("POT_CTL", "00000013"),
00397         ("POT_CTL", "00000015"),
00398         ("POT_CTL", "00000017"),
00399         ("POT_CTL", "00000019"),
00400         ("POT_CTL", "0000001B"),
00401     ]
00402     return self.ca.submitMessages(control_messages, " latchPots: ")
00403
00404 def initSensor(self):
00405     """
00406     Register sensor, set default timing settings
00407

```

```

00408     Returns:
00409         tuple (error string, response string) from final control message
00410     """
00411     logging.info(self.logininfo + "initSensor")
00412     if int(self.ca.FPGANum[7]) != self.ca.sensor.fpganumID:
00413         logging.error(
00414             self.logerr + "unable to confirm sensor compatibility with FPGA"
00415         )
00416     self.registers.update(self.ca.sensor.sens_registers)
00417     self.subregisters.extend(self.ca.sensor.sens_subregisters)
00418     for s in self.ca.sensor.sens_subregisters:
00419         sr = SubRegister(
00420             self,
00421             name=s[0].upper(),
00422             register=s[1].upper(),
00423             start_bit=s[2],
00424             width=s[3],
00425             writable=s[4],
00426         )
00427         setattr(self, s[0].upper(), sr)
00428         self.subreglist.append(s[0])
00429     self.ca.checkSensorVoltStat()
00430     control_messages = self.ca.sensorSpecific() + [
00431         # ring w/caps=01, relax=00, ring w/o caps = 02
00432         ("OSC_SELECT", "00"),
00433         ("FPA_DIVCLK_EN_ADDR", "00000001"), # TODO Make this a subregister
00434     ]
00435     return self.ca.submitMessages(control_messages, " initSensor: ")
00436
00437 def configADCs(self):
00438     """
00439     Sets default ADC configuration (does not latch settings)
00440
00441     Returns:
00442         tuple (error string, response string) from final control message
00443     """
00444     logging.info(self.logininfo + "configADCs")
00445
00446     control_messages = [
00447         # just in case ADC_RESET was set (pull all ADCs out # of reset)
00448         ("ADC_RESET", "00000000"),
00449         # workaround for uncertain behavior after previous readoff
00450         ("ADC1_CONFIG_DATA", "FFFFFFF"),
00451         ("ADC2_CONFIG_DATA", "FFFFFFF"),
00452         ("ADC3_CONFIG_DATA", "FFFFFFF"),
00453         ("ADC4_CONFIG_DATA", "FFFFFFF"),
00454         ("ADC_CTL", "FFFFFFF"),
00455         ("ADC1_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
00456         ("ADC2_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
00457         ("ADC3_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
00458         ("ADC4_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
00459         ("ADC5_CONFIG_DATA", "81A883FF"), # int Vref 2.50V
00460     ]
00461     return self.ca.submitMessages(control_messages, " configADCs: ")
00462
00463 def softReboot(self):
00464     """
00465     Perform software reboot of board. WARNING: board reboot will likely prevent
00466     correct communication responses and therefore will generate an error message
00467
00468     Returns:
00469         tuple (error string, response string) from final control message
00470     """
00471     logging.info(self.logininfo + "reboot")
00472     control_messages = [("RESET", "1")]
00473     return self.ca.submitMessages(control_messages, " disarm: ")
00474
00475 def disarm(self):
00476     """
00477     Takes camera out of trigger wait state. Has no effect if camera is not already
00478     in wait state.
00479
00480     Returns:
00481         tuple (error string, response string) from final control message
00482     """
00483     logging.info(self.logininfo + "disarm")
00484     self.ca.clearStatus()
00485     self.ca.armed = False
00486     control_messages = [
00487         ("HW_TRIG_EN", "0"),
00488         ("SW_TRIG_EN", "0"),

```

```

00489     ]
00490     return self.ca.submitMessages(control_messages, " disarm: ")
00491
00492 def startCapture(self, mode="Hardware"):
00493     """
00494     Selects trigger mode and enables board for image capture
00495
00496     Args:
00497         mode: trigger mode ("hardware"|"software"|"dual"|"h"|"s"|"d" , is case-
00498             insensitive)
00499
00500     Returns:
00501         tuple (error string, response string) from final control message
00502     """
00503     logging.info(self.logininfo + "startCapture")
00504     if self.ca.sensmanual:
00505         timingReg = "MANSHUT_MODE"
00506     else:
00507         timingReg = "HST_MODE"
00508
00509     if mode.upper()[0] == "S": # SOFTWARE
00510         trigmess = [
00511             ("HW_TRIG_EN", "0"),
00512             ("SW_TRIG_EN", "1"),
00513             ("SW_TRIG_START", "1"),
00514         ]
00515     else: # HARDWARE
00516         trigmess = [
00517             ("SW_TRIG_EN", "0"),
00518             ("HW_TRIG_EN", "1"),
00519         ]
00520
00521     control_messages = [
00522         ("ADC_CTL", "0000001F"), # configure all ADCs
00523         (timingReg, "1"),
00524     ]
00525
00526     control_messages.extend(trigmess)
00527     return self.ca.submitMessages(control_messages, " startCapture: ")
00528
00529 def readSRAM(self):
00530     """
00531     Start readoff of SRAM
00532
00533     Returns:
00534         tuple (error string, response string from register set)
00535     """
00536     logging.info(self.logininfo + "readSRAM")
00537     control_messages = [("READ_SRAM", "1")]
00538     return self.ca.submitMessages(control_messages, " readSRAM: ")
00539
00540 def waitForSRAM(self, timeout):
00541     """
00542     Wait until subreg 'SRAM_READY' flag is true or timeout is exceeded;
00543     timeout = None or zero means wait indefinitely
00544
00545     Args:
00546         timeout - time in seconds before readoff proceeds automatically without
00547             waiting for SRAM_READY flag
00548
00549     Returns:
00550         error string
00551     """
00552     logging.info(self.logininfo + "waitForSRAM")
00553     waiting = True
00554     starttime = time.time()
00555     err = ""
00556     while waiting:
00557         err, status = self.ca.getSubregister("SRAM_READY")
00558         if err:
00559             logging.error(
00560                 self.logerr + "error in register read: " + err + " (waitForSRAM)"
00561             )
00562         if int(status):
00563             waiting = False
00564             logging.info(self.logininfo + "SRAM ready")
00565         if self.ca.abort:
00566             waiting = False
00567             logging.info(self.logininfo + "readoff aborted by user")
00568             self.ca.abort = False
00569         if timeout and time.time() - starttime > timeout:

```

```

00570         err += self.logerr + "SRAM timeout; proceeding with download attempt"
00571         logging.error(err)
00572         return err
00573     return err
00574
00575     def getTimer(self):
00576         """
00577         Read value of on-board timer
00578
00579         Returns:
00580             timer value as integer
00581         """
00582         logging.info(self.logininfo + "getTimer")
00583         err, rval = self.ca.getRegister("TIMER_VALUE")
00584         if err:
00585             logging.error(
00586                 self.logerr + "unable to retrieve timer information (getTimer), "
00587                 'returning "0" '
00588             )
00589             return 0
00590         return int(rval, 16)
00591
00592     def resetTimer(self):
00593         """
00594         Reset on-board timer
00595
00596         Returns:
00597             tuple (error string, response string from register set)
00598         """
00599         logging.info(self.logininfo + "resetTimer")
00600         control_messages = [("RESET_TIMER", "1"), ("RESET_TIMER", "0")]
00601         return self.ca.submitMessages(control_messages, " resetTimer: ")
00602
00603     def enableLED(self, status):
00604         """
00605         Enable/disable on-board LEDs
00606
00607         Args:
00608             status: 0 for disabled, 1 for enabled
00609
00610         Returns:
00611             tuple: (error string, response string from setSubregister())
00612         """
00613         logging.info(self.logininfo + "enableLED")
00614         if status:
00615             status = 1
00616         return self.ca.setSubregister("LED_EN", str(status))
00617
00618     def setLED(self, LED, status):
00619         """
00620         Illuminate on-board LED
00621
00622         Args:
00623             LED: LED number (1-8)
00624             status: 0 is off, 1 is on
00625
00626         Returns:
00627             tuple: (error string, response string from setSubregister())
00628         """
00629         logging.info(self.logininfo + "setLED")
00630         key = "LED" + str(LED)
00631         return self.ca.setSubregister(key, str(status))
00632
00633     def setPowerSave(self, status):
00634         """
00635         Select powersave option
00636
00637         Args:
00638             status: setting for powersave option (1 is enabled)
00639
00640         Returns:
00641             tuple (error string, response string from setSubregister())
00642         """
00643         logging.info(self.logininfo + "setPowerSave")
00644         if status:
00645             status = 1
00646         return self.ca.setSubregister("POWERSAVE", str(status))
00647
00648     def setPPER(self, pollperiod):
00649         """
00650         Set polling period for ADCs.

```

```

00651     Args:
00652         pollperiod: milliseconds, between 1 and 255, defaults to 50
00653
00654     Returns:
00655         tuple (error string, response string from setSubregister OR invalid time
00656             setting string)
00657     """
00658     logging.debug(self.logdebug + "setPPER: time = " + str(pollperiod))
00659     if pollperiod is None:
00660         pollperiod = 50
00661     if not isinstance(pollperiod, int) or pollperiod < 1 or pollperiod > 255:
00662         err = (
00663             self.logerr + "invalid poll period submitted. Setting remains "
00664             "unchanged. "
00665         )
00666         logging.error(err)
00667         return err, str(pollperiod)
00668     else:
00669         binset = bin(pollperiod)[2:].zfill(8)
00670         return self.ca.setSubregister("PPER", binset)
00671
00672 def getTemp(self, scale=None, offset=None, slope=None):
00673     """
00674     Read temperature sensor
00675     Args:
00676         scale: temperature scale to report (defaults to C, options are F and K)
00677         offset: unused
00678         slope: unused
00679     Returns:
00680         temperature as float on given scale
00681     """
00682     logging.debug(self.logdebug + "getTemp: scale = " + str(scale))
00683     err, rval = self.ca.getRegister("TEMP_SENSE_DATA")
00684     if err:
00685         logging.error(
00686             self.logerr + "unable to retrieve temperature information ("
00687             'getTemp), returning "0" '
00688         )
00689         return 0.0
00690
00691     ctemp = int(rval[-3:], 16) / 16.0
00692     if scale == "K":
00693         temp = ctemp + 273.15
00694     elif scale == "F":
00695         temp = 1.8 * ctemp + 32
00696     else:
00697         temp = ctemp
00698     return temp
00699
00700 def getPressure(self, offset, sensitivity, units):
00701     """
00702     Read pressure sensor
00703
00704     Currently unimplemented
00705
00706     Returns:
00707         0 as float
00708     """
00709     logging.warning(
00710         "WARNING: [LLNL_v1] 'getPressure' is not implemented on the LLNLv1 board"
00711     )
00712     return 0.0
00713
00714 def clearStatus(self):
00715     """
00716     Check status registers to clear them
00717
00718     Returns:
00719         error string
00720     """
00721     logging.debug(self.logdebug + "clearStatus")
00722     err1, rval = self.ca.getRegister("STAT_REG_SRC")
00723     err2, rval = self.ca.getRegister("STAT_REG2_SRC")
00724     err = err1 + err2
00725     if err:
00726         logging.error(self.logerr + "clearStatus failed")
00727     return err
00728
00729 def checkStatus(self):
00730     """
00731     Check status register, convert to reverse-order bit stream (i.e., bit 0 is

```

```

00732         statusbits[0])
00733
00734     Returns:
00735         bit string (no '0b') in reversed order
00736     """
00737     logging.debug(self.logdebug + "checkStatus")
00738     err, rval = self.ca.getRegister("STAT_REG")
00739     if not rval:
00740         logging.error(
00741             self.logerr + "Unable to check status register (zeroes returned)"
00742         )
00743         rval = "0"
00744     rvalbits = bin(int(rval, 16))[2:].zfill(32)
00745     statusbits = rvalbits[::-1]
00746     return statusbits # TODO: add error handling
00747
00748 def checkStatus2(self):
00749     """
00750     Check second status register, convert to reverse-order bit stream (i.e., bit 0
00751     is statusbits[0])
00752
00753     Returns: bit string (no '0b') in reversed order
00754     """
00755     logging.debug(self.logdebug + "checkStatus2")
00756     err, rval = self.ca.getRegister("STAT_REG2")
00757     if not rval:
00758         logging.error(
00759             self.logerr + "Unable to check status register 2 (zeroes returned)"
00760         )
00761         rval = "0"
00762     rvalbits = bin(int(rval, 16))[2:].zfill(5)
00763     statusbits = rvalbits[::-1]
00764     return statusbits # TODO: add error handling
00765
00766 def reportStatus(self):
00767     """
00768     Check contents of status register, print relevant messages
00769     """
00770     statusbits = self.checkStatus()
00771     statusbits2 = self.checkStatus2()
00772     logging.info(self.loginfo + "Status report:")
00773     if int(statusbits[0]):
00774         print(self.loginfo + "Sensor read complete")
00775     if int(statusbits[1]):
00776         print(self.loginfo + "Coarse trigger detected")
00777     if int(statusbits[2]):
00778         print(self.loginfo + "Fine trigger detected")
00779     if int(statusbits[5]):
00780         print(self.loginfo + "Sensor readout in progress")
00781     if int(statusbits[6]):
00782         print(self.loginfo + "Sensor readout complete")
00783     if int(statusbits[7]):
00784         print(self.loginfo + "SRAM readout started")
00785     if int(statusbits[8]):
00786         print(self.loginfo + "SRAM readout complete")
00787     if int(statusbits[9]):
00788         print(self.loginfo + "High-speed timing configuration started")
00789     if int(statusbits[10]):
00790         print(self.loginfo + "All ADCs configured")
00791     if int(statusbits[11]):
00792         print(self.loginfo + "All pots configured")
00793     if int(statusbits[13]):
00794         print(self.loginfo + "Timer has reset")
00795     if int(statusbits[14]):
00796         print(self.loginfo + "Camera is Armed")
00797     self.ca.sensor.reportStatusSensor(statusbits, statusbits2)
00798     temp = int(statusbits[27:15:-1], 2) / 16.0
00799     logging.info(
00800         self.loginfo + "Temperature reading: " + "{0:1.2f}".format(temp) + " C"
00801     )
00802     # press = int(statusbits[:27:-1], 2)
00803     # logging.info(self.loginfo + "Pressure reading: " + "{0:1.2f}".format(press))
00804     if int(statusbits2[0]):
00805         print(self.loginfo + "FPA_IF_TO")
00806     if int(statusbits2[1]):
00807         print(self.loginfo + "SRAM_RO_TO")
00808     if int(statusbits2[2]):
00809         print(self.loginfo + "PixelRd Timeout Error")
00810     if int(statusbits2[3]):
00811         print(self.loginfo + "UART_TX_TO_RST")
00812     if int(statusbits2[4]):

```

```

00813         print(self.loginfo + "UART_RX_TO_RST")
00814
00815     def reportEdgeDetects(self):
00816         """
00817         Unimplemented
00818         """
00819         logging.warning(
00820             self.logwarn + "'reportEdgeDetects' is not implemented on the LLNLv1 "
00821             "board "
00822         )
00823
00824     def dumpStatus(self):
00825         """
00826         Create dictionary of status values, DAC settings, monitor values, and register
00827         values
00828
00829         WARNING: the behavior of self-resetting subregisters may be difficult to predict
00830         and may generate contradictory results
00831
00832         Returns:
00833             dictionary of system diagnostic values
00834         """
00835         statusbits = self.checkStatus()
00836         statusbits2 = self.checkStatus2()
00837         temp = self.ca.getTemp()
00838
00839         statDict = OrderedDict(
00840             {
00841                 "Temperature reading": "{0:1.2f}".format(temp) + " C",
00842                 "Sensor read complete": str(statusbits[0]),
00843                 "Coarse trigger detected": str(statusbits[1]),
00844                 "Fine trigger detected": str(statusbits[2]),
00845                 "Sensor readout in progress": str(statusbits[5]),
00846                 "Sensor readout complete": str(statusbits[6]),
00847                 "SRAM readout started": str(statusbits[7]),
00848                 "SRAM readout complete": str(statusbits[8]),
00849                 "High-speed timing configured": str(statusbits[9]),
00850                 "All ADCs configured": str(statusbits[10]),
00851                 "All pots configured": str(statusbits[11]),
00852                 "HST_All_W_En detected": str(statusbits[12]),
00853                 "Timer has reset": str(statusbits[13]),
00854                 "Camera is Armed": str(statusbits[14]),
00855                 "FPA_IF_TO": str(statusbits2[0]),
00856                 "SRAM_RO_TO": str(statusbits2[1]),
00857                 "PixelRd Timeout Error": str(statusbits2[2]),
00858                 "UART_TX_TO_RST": str(statusbits2[3]),
00859                 "UART_RX_TO_RST": str(statusbits2[4]),
00860             }
00861         )
00862
00863         POTDict = OrderedDict()
00864         MonDict = OrderedDict()
00865         for entry in self.subreg_aliases:
00866             if self.subreg_aliases[entry][0] == "P":
00867                 val = str(round(self.ca.getPotV(entry), 3)) + " V"
00868                 POTDict["POT_" + entry] = val
00869             else:
00870                 val = str(round(self.ca.getMonV(entry), 3)) + " V"
00871                 MonDict[entry] = val
00872
00873         regDict = OrderedDict()
00874         for key in self.registers.keys():
00875             err, rval = self.ca.getRegister(key)
00876             regDict[key] = rval
00877
00878         dumpDict = OrderedDict()
00879         for x in [statDict, MonDict, POTDict, regDict]:
00880             dumpDict.update(x)
00881         return dumpDict
00882
00883     """
00884     """
00885     Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00886     LLNL-CODE-838080
00887
00888     This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00889     contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00890     and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00891     'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00892     be made under this license.
00893     """

```


7.13 C:/Users/hill35/git/camera_python/nsCamera/boards/LLNL_v4.py File Reference

Classes

- class `nsCamera.boards.LLNL_v4.llnl_v4`

Namespaces

- namespace `nsCamera`
- namespace `nsCamera.boards`
- namespace `nsCamera.boards.LLNL_v4`

7.14 LLNL_v4.py

[Go to the documentation of this file.](#)

```
00001 # -*- coding: utf-8 -*-
00002 """
00003 LLNLv4 board definition, including monitors, DACS, and other board-specific settings
00004
00005 Author: Jeremy Martin Hill (jerhill@llnl.gov)
00006 Author: Matthew Dayton (dayton5@llnl.gov)
00007
00008 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00009 LLNL-CODE-838080
00010
00011 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00012 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00013 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00014 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00015 be made under this license.
00016
00017 Version: 2.1.2 (February 2025)
00018 """
00019
00020 import logging
00021 import string
00022 import time
00023 from collections import OrderedDict
00024
00025 from nsCamera.utils.Packet import Packet
00026 from nsCamera.utils.Subregister import SubRegister
00027
00028
00029 class llnl_v4:
00030     """
00031     Livermore LLNL v4.0 board
00032
00033     Compatible communication protocols: RS422, GigE
00034     Compatible sensors: icarus, icarus2, daedalus
00035     """
00036
00037     # FPGA register map - use '.upper()' on keys to ensure uppercase lookup
00038     registers = OrderedDict(
00039         {
00040             "FPGA_NUM": "000",
00041             "FPGA_REV": "001",
00042             "HS_TIMING_CTL": "010",
00043             "HS_TIMING_DATA_ALO": "013",
00044             "HS_TIMING_DATA_AHI": "014",
00045             "HS_TIMING_DATA_BLO": "015",
00046             "HS_TIMING_DATA_BHI": "016",
00047             "SW_TRIGGER_CONTROL": "017",
00048             "SW_COARSE_CONTROL": "01C",
00049             "STAT_REG": "024",
```

```

00050         "CTRL_REG": "025",
00051         "DAC_CTL": "026",
00052         "DAC_REG_A_AND_B": "027",
00053         "DAC_REG_C_AND_D": "028",
00054         "DAC_REG_E_AND_F": "029",
00055         "DAC_REG_G_AND_H": "02A",
00056         "SW_RESET": "02D",
00057         "HST_SETTINGS": "02E",
00058         "STAT_REG_SRC": "02F",
00059         "STAT_REG2": "030",
00060         "STAT_REG2_SRC": "031",
00061         "ADC_BYTECOUNTER": "032",
00062         "RBP_PIXEL_CNTR": "033",
00063         "DIAG_MAX_CNT_0": "034",
00064         "DIAG_MAX_CNT_1": "035",
00065         "DIAG_CNTR_VAL_0": "036",
00066         "DIAG_CNTR_VAL_1": "037",
00067         "STAT_EDGE_DETECTS": "038",
00068         "TRIGGER_CTL": "03A",
00069         "SRAM_CTL": "03B",
00070         "TIMER_CTL": "03C",
00071         "TIMER_VALUE": "03D",
00072         "HSTALLWEN_WAIT_TIME": "03F",
00073         "FPA_ROW_INITIAL": "042",
00074         "FPA_ROW_FINAL": "043",
00075         "FPA_FRAME_INITIAL": "044",
00076         "FPA_FRAME_FINAL": "045",
00077         "FPA_DIVCLK_EN_ADDR": "046",
00078         "FPA_OSCILLATOR_SEL_ADDR": "047",
00079         "SUSPEND_TIME": "04D",
00080         "FPA_INTERFACE_STATE": "04E",
00081         "DELAY_READOFF": "04F",
00082         "STAT_REG_SEC": "060",
00083         "ADC_CTL": "090",
00084         "ADC1_CONFIG_DATA": "091",
00085         "ADC2_CONFIG_DATA": "092",
00086         "ADC3_CONFIG_DATA": "093",
00087         "ADC4_CONFIG_DATA": "094",
00088         "ADC5_DATA_1": "095",
00089         "ADC5_DATA_2": "096",
00090         "ADC5_DATA_3": "097",
00091         "ADC5_DATA_4": "098",
00092         "ADC6_DATA_1": "099",
00093         "ADC6_DATA_2": "09A",
00094         "ADC6_DATA_3": "09B",
00095         "ADC6_DATA_4": "09C",
00096         "ADC_PPER": "09D",
00097         "ADC_RESET": "09E",
00098     }
00099 )
00100
00101 subregisters = [
00102
00104     ("HST_MODE", "HS_TIMING_CTL", 0, 1, True),
00105     ("SW_TRIG_START", "SW_TRIGGER_CONTROL", 0, 1, True),
00106     ("SW_COARSE_TRIGGER", "SW_COARSE_CONTROL", 0, 1, True),
00107     ("LED_EN", "CTRL_REG", 1, 1, True),
00108     ("COLQUENCHEN", "CTRL_REG", 2, 1, True),
00109     ("POWERSAVE", "CTRL_REG", 3, 1, True),
00110     ("PDBIAS_LOW", "CTRL_REG", 6, 1, True),
00111     # ("SWACK", "CTRL_REG", 10, 1, True),
00112     ("DACA", "DAC_REG_A_AND_B", 31, 16, True),
00113     ("DACB", "DAC_REG_A_AND_B", 15, 16, True),
00114     ("DACC", "DAC_REG_C_AND_D", 31, 16, True),
00115     ("DACD", "DAC_REG_C_AND_D", 15, 16, True),
00116     ("DACE", "DAC_REG_E_AND_F", 31, 16, True),
00117     ("DACF", "DAC_REG_E_AND_F", 15, 16, True),
00118     ("DACG", "DAC_REG_G_AND_H", 31, 16, True),
00119     ("DACH", "DAC_REG_G_AND_H", 15, 16, True),
00120     ("RESET", "SW_RESET", 0, 1, True),
00121     ("HST_SW_CTL_EN", "HST_SETTINGS", 0, 1, True),
00122     ("SW_HSTALLWEN", "HST_SETTINGS", 1, 1, True),
00123     ("MAXERR_FIT", "DIAG_MAX_CNT_0", 31, 16, True),
00124     ("MAXERR_SRT", "DIAG_MAX_CNT_0", 7, 8, True),
00125     ("MAXERR_UTTR", "DIAG_MAX_CNT_1", 31, 16, True),
00126     ("MAXERR_URTR", "DIAG_MAX_CNT_1", 15, 16, True),
00127     ("HW_TRIG_EN", "TRIGGER_CTL", 0, 1, True),
00128     ("SW_TRIG_EN", "TRIGGER_CTL", 2, 1, True),
00129     ("READOFF_DELAY_EN", "TRIGGER_CTL", 4, 1, True),
00130     ("READ_SRAM", "SRAM_CTL", 0, 1, True),
00131     ("RESET_TIMER", "TIMER_CTL", 0, 1, True),

```

```

00132         ("OSC_SELECT", "FPA_OSCILLATOR_SEL_ADDR", 1, 2, True),
00133         ("PPER", "ADC_PPER", 7, 8, True),
00134
00137         ("SRAM_READY", "STAT_REG", 0, 1, False),
00138         ("STAT_COARSE", "STAT_REG", 1, 1, False),
00139         ("STAT_FINE", "STAT_REG", 2, 1, False),
00140         ("STAT_SENSREADIP", "STAT_REG", 5, 1, False),
00141         ("STAT_SENSREADDONE", "STAT_REG", 6, 1, False),
00142         ("STAT_SRAMREADSTART", "STAT_REG", 7, 1, False),
00143         ("STAT_SRAMREADDONE", "STAT_REG", 8, 1, False),
00144         ("STAT_HSTCONFIGSTART", "STAT_REG", 9, 1, False),
00145         ("STAT_ADCSCONFIGURED", "STAT_REG", 10, 1, False),
00146         ("STAT_DACSCONFIGURED", "STAT_REG", 11, 1, False),
00147         ("STAT_TIMERCOUNTERRESET", "STAT_REG", 13, 1, False),
00148         ("STAT_HSTCONFIGDONE", "STAT_REG", 16, 1, False),
00149         ("STAT_ARMED", "STAT_REG", 14, 1, False),
00150         ("STAT_TEMP", "STAT_REG", 23, 7, False),
00151         ("STAT_PRESS", "STAT_REG", 31, 8, False),
00152         ("FPA_IF_TO", "STAT_REG2", 0, 1, False),
00153         ("SRAM_RO_TO", "STAT_REG2", 1, 1, False),
00154         ("PIXELRD_TOUT_ERR", "STAT_REG2", 2, 1, False),
00155         ("UART_TX_TO_RST", "STAT_REG2", 3, 1, False),
00156         ("UART_RX_TO_RST", "STAT_REG2", 4, 1, False),
00157         ("FIT_COUNT", "DIAG_CNTR_VAL_0", 31, 16, False),
00158         ("SRT_COUNT", "DIAG_CNTR_VAL_0", 7, 8, False),
00159         ("UTTR_COUNT", "DIAG_CNTR_VAL_1", 31, 16, False),
00160         ("URTR_COUNT", "DIAG_CNTR_VAL_1", 15, 16, False),
00161         # monitor ADC channels defined here - the poll period will need to be set
00162         # during camera initialization (x98)
00163         ("MON_CH1", "ADC5_DATA_1", 11, 12, False),
00164         ("MON_CH2", "ADC5_DATA_1", 23, 12, False),
00165         ("MON_CH3", "ADC5_DATA_2", 11, 12, False),
00166         ("MON_CH4", "ADC5_DATA_2", 23, 12, False),
00167         ("MON_CH5", "ADC5_DATA_3", 11, 12, False),
00168         ("MON_CH6", "ADC5_DATA_3", 23, 12, False),
00169         ("MON_CH7", "ADC5_DATA_4", 11, 12, False),
00170         ("MON_CH8", "ADC5_DATA_4", 23, 12, False),
00171         ("MON_CH9", "ADC6_DATA_1", 11, 12, False),
00172         ("MON_CH10", "ADC6_DATA_1", 23, 12, False),
00173         ("MON_CH11", "ADC6_DATA_2", 11, 12, False),
00174         ("MON_CH12", "ADC6_DATA_2", 23, 12, False),
00175         ("MON_CH13", "ADC6_DATA_3", 11, 12, False),
00176         ("MON_CH14", "ADC6_DATA_3", 23, 12, False),
00177         ("MON_CH15", "ADC6_DATA_4", 11, 12, False),
00178         ("MON_CH16", "ADC6_DATA_4", 23, 12, False),
00179     ]
00180
00181     def __init__(self, camassem):
00182         self.ca = camassem
00183         self.logcrit = self.ca.logcritbase + "[LLNL_v4] "
00184         self.logerr = self.ca.logerrbase + "[LLNL_v4] "
00185         self.logwarn = self.ca.logwarnbase + "[LLNL_v4] "
00186         self.loginfo = self.ca.loginfobase + "[LLNL_v4] "
00187         self.logdebug = self.ca.logdebugbase + "[LLNL_v4] "
00188         logging.info(self.loginfo + "Initializing board object")
00189         self.VREF = 3.3 # must be supplied externally for ADC128S102
00190         self.ADC5_mult = 1
00191
00192         # ADC128S102; False => monitor range runs 0 to monmax, True => +/- monmax
00193         self.ADC5_bipolar = False
00194         self.rs422_baud = 921600
00195         self.rs422_cmd_wait = 0.3
00196
00197         fpgaNum_pkt = Packet(cmd="1", addr=self.registers["FPGA_NUM"])
00198         fpgaRev_pkt = Packet(cmd="1", addr=self.registers["FPGA_REV"])
00199
00200         _, _ = self.ca.sendCMD(fpgaNum_pkt) # dummy duplicate call
00201         err, rval = self.ca.sendCMD(fpgaRev_pkt)
00202         self.ca.FPGA_NUM = rval[8:16]
00203
00204         err, rval = self.ca.sendCMD(fpgaRev_pkt)
00205         self.ca.FPGA_Version = rval[8:16]
00206
00207         self.defoff = 34.5 # default pressure sensor offset
00208         self.defsens = 92.5 # default pressure sensor sensitivity
00209
00210         # TODO: move to sensor scripts?
00211         # map channels to signal names for abstraction at the camera assembler level;
00212         # each requires a corresponding entry in 'subregisters'
00213         self.icarus_subreg_aliases = OrderedDict(
00214             {

```

```

00215         "HST_A_PDELAY": "DACA",
00216         "HST_A_NDELAY": "DACB",
00217         "HST_B_PDELAY": "DACC",
00218         "HST_B_NDELAY": "DACD",
00219         "HST_RO_IBIAS": "DACE",
00220         "HST_RO_NC_IBIAS": "DACE",
00221         "HST_OSC_CTL": "DACF",
00222         "VAB": "DACG",
00223         "VRST": "DACH",
00224         "MON_PRES_MINUS": "MON_CH1",
00225         "MON_PRES_PLUS": "MON_CH2",
00226         "MON_TEMP": "MON_CH3",
00227         "MON_COL_TOP_IBIAS_IN": "MON_CH4",
00228         "MON_HST_OSC_R_IBIAS": "MON_CH5",
00229         "MON_VAB": "MON_CH6",
00230         "MON_HST_RO_IBIAS": "MON_CH7",
00231         "MON_HST_RO_NC_IBIAS": "MON_CH7",
00232         "MON_VRST": "MON_CH8",
00233         "MON_COL_BOT_IBIAS_IN": "MON_CH9",
00234         "MON_HST_A_PDELAY": "MON_CH10",
00235         "MON_HST_B_NDELAY": "MON_CH11",
00236         "DOSIMETER": "MON_CH12",
00237         "MON_HST_OSC_VREF_IN": "MON_CH13",
00238         "MON_HST_B_PDELAY": "MON_CH14",
00239         "MON_HST_OSC_CTL": "MON_CH15",
00240         "MON_HST_A_NDELAY": "MON_CH16",
00241         "MON_CHA": "MON_CH10",
00242         "MON_CHB": "MON_CH16",
00243         "MON_CHC": "MON_CH14",
00244         "MON_CHD": "MON_CH11",
00245         "MON_CHE": "MON_CH7",
00246         "MON_CHE": "MON_CH15",
00247         "MON_CHG": "MON_CH6",
00248         "MON_CHH": "MON_CH8",
00249     }
00250 )
00251 # Read-only; identifies controls corresponding to monitors
00252 self.icarus_monitor_controls = OrderedDict(
00253     {
00254         "MON_CH10": "DACA",
00255         "MON_CH16": "DACB",
00256         "MON_CH14": "DACC",
00257         "MON_CH11": "DACD",
00258         "MON_CH7": "DACE",
00259         "MON_CH15": "DACF",
00260         "MON_CH6": "DACG",
00261         "MON_CH8": "DACH",
00262     }
00263 )
00264 self.daedalus_subreg_aliases = OrderedDict(
00265     {
00266         "HST_OSC_VREF_IN": "DACC",
00267         "HST_OSC_CTL": "DACE",
00268         "COL_TST_IN": "DACF",
00269         "VAB": "DACG",
00270         "VRST": "DACH",
00271         "MON_PRES_MINUS": "MON_CH1",
00272         "MON_PRES_PLUS": "MON_CH2",
00273         "MON_TEMP": "MON_CH3",
00274         "MON_VAB": "MON_CH6",
00275         "MON_HST_OSC_CTL": "MON_CH7",
00276         "MON_TSENSE_OUT": "MON_CH10",
00277         "MON_BGREF": "MON_CH11",
00278         "DOSIMETER": "MON_CH12",
00279         "MON_HST_RO_NC_IBIAS": "MON_CH13",
00280         "MON_HST_OSC_VREF_IN": "MON_CH14",
00281         "MON_COL_TST_IN": "MON_CH15",
00282         "MON_HST_OSC_PBIAS_PAD": "MON_CH16",
00283         "MON_CHC": "MON_CH14",
00284         "MON_CHE": "MON_CH7",
00285         "MON_CHE": "MON_CH15",
00286         "MON_CHG": "MON_CH6",
00287         "MON_CHH": "MON_CH8",
00288     }
00289 )
00290 # Read-only; identifies controls corresponding to monitors
00291 self.daedalus_monitor_controls = OrderedDict(
00292     {
00293         "MON_CH14": "DACC",
00294         "MON_CH7": "DACE",
00295         "MON_CH15": "DACF",

```

```

00296         "MON_CH6": "DACG",
00297         "MON_CH8": "DACH",
00298     }
00299 )
00300 self.subreglist = []
00301 for s in self.subregisters:
00302     self.subreglist.append(s[0].upper())
00303     sr = SubRegister(
00304         self,
00305         name=s[0].upper(),
00306         register=s[1].upper(),
00307         start_bit=s[2],
00308         width=s[3],
00309         writable=s[4],
00310     )
00311     setattr(self, s[0].upper(), sr)
00312
00313 # set voltage ranges for all DACs - WARNING: actual output voltage limited to
00314 # external supply (3.3 V)
00315 # setpot('potx', n) will generate 3.3 V for all n > .66
00316 for n in range(0, 8):
00317     potname = "DAC" + string.ascii_uppercase[n]
00318     potobj = getattr(self, potname)
00319     potobj.minV = 0
00320     potobj.maxV = 5 #
00321     potobj.resolution = (
00322         1.0 * potobj.maxV - potobj.minV
00323     ) / potobj.max_value # 76 uV / LSB
00324
00325 def initBoard(self):
00326     """
00327     Register and reset board, set up firmware for sensor
00328
00329     Returns:
00330         tuple (error string, response string) from final control message
00331     """
00332     logging.info(self.logininfo + "initBoard LLNLv4")
00333     control_messages = []
00334     self.clearStatus()
00335     self.configADCs()
00336     return self.ca.submitMessages(control_messages, " initBoard: ")
00337
00338 def initPots(self):
00339     """
00340     Dummy function; initial DAC values are set by firmware at startup
00341
00342     Returns:
00343         tuple (empty string, empty string)
00344     """
00345     logging.debug(self.logdebug + "InitPots")
00346     return "", ""
00347
00348 def latchPots(self):
00349     """
00350     Latch DAC settings into sensor
00351
00352     Returns:
00353         tuple (error string, response string) from final control message
00354     """
00355     logging.info(self.logininfo + "latchPots")
00356     control_messages = [
00357         ("DAC_CTL", "00000001"), # latches register settings for DACA
00358         ("DAC_CTL", "00000003"),
00359         ("DAC_CTL", "00000005"),
00360         ("DAC_CTL", "00000007"),
00361         ("DAC_CTL", "00000009"),
00362         ("DAC_CTL", "0000000B"),
00363         ("DAC_CTL", "0000000D"),
00364         ("DAC_CTL", "0000000F"),
00365     ]
00366     return self.ca.submitMessages(control_messages, " latchPots: ")
00367
00368 def initSensor(self):
00369     """
00370     Register sensor, set default timing settings
00371
00372     Returns:
00373         tuple (error string, response string) from final control message
00374     """
00375     logging.info(self.logininfo + "initSensor")
00376     if int(self.ca.FPGANum[7]) != self.ca.sensor.fpganumID:

```

```

00377         logging.warning(
00378             self.logwarn + "unable to confirm sensor compatibility with FPGA"
00379         )
00380     self.registers.update(self.ca.sensor.sens_registers)
00381     self.subregisters.extend(self.ca.sensor.sens_subregisters)
00382     for s in self.ca.sensor.sens_subregisters:
00383         sr = SubRegister(
00384             self,
00385             name=s[0].upper(),
00386             register=s[1].upper(),
00387             start_bit=s[2],
00388             width=s[3],
00389             writable=s[4],
00390         )
00391         setattr(self, s[0].upper(), sr)
00392         self.subreglist.append(s[0])
00393     # TODO: self.ca.checkSensorVoltStat() # SENSOR_VOLT_STAT and SENSOR_VOLT_CTL are
00394     # deactivated for v4 icarus and daedalus firmware for now, is this permanent?
00395     control_messages = self.ca.sensorSpecific() + [
00396         # ring w/caps=01, relax=00, ring w/o caps = 02
00397         ("OSC_SELECT", "00"),
00398         ("FPA_DIVCLK_EN_ADDR", "00000001"),
00399     ]
00400     return self.ca.submitMessages(control_messages, " initSensor: ")
00401
00402 def configADCs(self):
00403     """
00404     Sets default ADC configuration (does not latch settings)
00405
00406     Returns:
00407         tuple (error string, response string) from final control message
00408     """
00409     logging.info(self.loginfo + "configADCs")
00410
00411     control_messages = [
00412         # just in case ADC_RESET was set on any of the ADCs (pull all ADCs out of
00413         # reset)
00414         ("ADC_RESET", "00000000"),
00415         # workaround for uncertain behavior after previous readoff
00416         ("ADC1_CONFIG_DATA", "FFFFFFFF"),
00417         ("ADC2_CONFIG_DATA", "FFFFFFFF"),
00418         ("ADC3_CONFIG_DATA", "FFFFFFFF"),
00419         ("ADC4_CONFIG_DATA", "FFFFFFFF"),
00420         ("ADC_CTL", "FFFFFFFF"),
00421         ("ADC1_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
00422         ("ADC2_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
00423         ("ADC3_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
00424         ("ADC4_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
00425     ]
00426     return self.ca.submitMessages(control_messages, " configADCs: ")
00427
00428 def softReboot(self):
00429     """
00430     Perform software reboot of board. WARNING: board reboot will likely prevent
00431     correct response and therefore will generate an error message
00432
00433     Returns:
00434         tuple (error string, response string) from final control message
00435     """
00436     logging.info(self.loginfo + "reboot")
00437     control_messages = [("RESET", "0")]
00438     return self.ca.submitMessages(control_messages, " disarm: ")
00439
00440 def disarm(self):
00441     """
00442     Takes camera out of trigger wait state. Has no effect if camera is not in wait
00443     state.
00444
00445     Returns:
00446         tuple (error string, response string) from final control message
00447     """
00448     logging.info(self.loginfo + "disarm")
00449     self.ca.clearStatus()
00450     self.ca.armed = False
00451     control_messages = [
00452         ("HW_TRIG_EN", "0"),
00453         ("SW_TRIG_EN", "0"),
00454     ]
00455     self.ca.comms.skipError = False
00456     return self.ca.submitMessages(control_messages, " disarm: ")
00457

```

```

00458     def startCapture(self, mode="Hardware"):
00459         """
00460         Selects trigger mode and enables board for image capture
00461
00462         Args:
00463             mode: trigger mode ("hardware"|"software"|"dual"|"h"|"s"|"d" , is case-
00464                 insensitive)
00465
00466         Returns:
00467             tuple (error string, response string) from final control message
00468         """
00469         logging.info(self.logininfo + "startCapture")
00470         if self.ca.sensmanual:
00471             timingReg = "MANSHUT_MODE"
00472         else:
00473             timingReg = "HST_MODE"
00474
00475         if mode.upper()[0] == "S": # SOFTWARE
00476             trigmess = [
00477                 ("HW_TRIG_EN", "0"),
00478                 ("SW_TRIG_EN", "1"),
00479                 ("SW_TRIG_START", "1"),
00480             ]
00481         else: # HARDWARE
00482             trigmess = [
00483                 ("SW_TRIG_EN", "0"),
00484                 ("HW_TRIG_EN", "1"),
00485             ]
00486
00487         control_messages = [
00488             ("ADC_CTL", "0000000F"), # configure all ADCs
00489             (timingReg, "1"),
00490         ]
00491
00492         control_messages.extend(trigmess)
00493         return self.ca.submitMessages(control_messages, " startCapture: ")
00494
00495     def readSRAM(self):
00496         """
00497         Start readoff of SRAM
00498
00499         Returns:
00500             tuple (error string, response string from register set)
00501         """
00502         logging.info(self.logininfo + "readSRAM")
00503         control_messages = [("READ_SRAM", "1")]
00504         return self.ca.submitMessages(control_messages, " readSRAM: ")
00505
00506     def waitForSRAM(self, timeout):
00507         """
00508         Wait until subreg 'SRAM_READY' flag is true or timeout is exceeded;
00509         timeout = None or zero means wait indefinitely
00510
00511         Args:
00512             timeout - time in seconds before readoff proceeds automatically without
00513                     waiting for SRAM_READY flag
00514
00515         Returns:
00516             error string
00517         """
00518         logging.info(self.logininfo + "waitForSRAM, timeout = " + str(timeout))
00519         waiting = True
00520         starttime = time.time()
00521         err = ""
00522         while waiting:
00523             err, status = self.ca.getSubregister("SRAM_READY")
00524             if err:
00525                 err = self.logerr + "error in register read: " + err + " (waitForSRAM)"
00526                 logging.error(err)
00527             if int(status):
00528                 waiting = False
00529                 logging.info(self.logininfo + "SRAM ready")
00530             if self.ca.abort:
00531                 waiting = False
00532                 logging.info(self.logininfo + "readoff aborted by user")
00533                 self.ca.abort = False
00534             if timeout and time.time() - starttime > timeout:
00535                 err += self.logerr + "SRAM timeout; proceeding with download attempt"
00536                 logging.error(err)
00537                 return err
00538         # Slow down for debugging (avoid thousands of messages)

```

```

00539         if self.ca.verbose >= 5:
00540             time.sleep(0.5)
00541     return err
00542
00543 def getTimer(self):
00544     """
00545     Read value of on-board timer
00546
00547     Returns:
00548         timer value as integer
00549     """
00550     logging.info(self.loginfo + "getTimer")
00551     err, rval = self.ca.getRegister("TIMER_VALUE")
00552     if err:
00553         logging.error(
00554             self.logerr + "unable to retrieve timer information (getTimer), "
00555             'returning "0" '
00556         )
00557     return 0
00558     return int(rval, 16)
00559
00560 def resetTimer(self):
00561     """
00562     Reset on-board timer
00563
00564     Returns:
00565         tuple (error string, response string from register set)
00566     """
00567     logging.info(self.loginfo + "resetTimer")
00568     control_messages = [{"RESET_TIMER", "1"}, {"RESET_TIMER", "0"}]
00569     return self.ca.submitMessages(control_messages, " resetTimer: ")
00570
00571 def enableLED(self, status):
00572     """
00573     Dummy function; feature is not implemented on LLNL_V4 board
00574
00575     Returns:
00576         tuple: dummy of (error string, response string from setSubregister())
00577     """
00578     del status
00579     return "", "0"
00580
00581 def setLED(self, LED, status):
00582     """
00583     Dummy function; feature is not implemented on LLNL_V4 board
00584
00585     Returns:
00586         tuple: dummy of (error string, response string from setSubregister())
00587     """
00588     del LED, status
00589     return "", "0"
00590
00591 def setPowerSave(self, status):
00592     """
00593     Select powersave option
00594
00595     Args:
00596         status: setting for powersave option (1 is enabled)
00597
00598     Returns:
00599         tuple (error string, response string from setSubregister())
00600     """
00601     if status:
00602         status = 1
00603     return self.ca.setSubregister("POWERSAVE", str(status))
00604
00605 def setPPER(self, pollperiod):
00606     """
00607     Set polling period for ADCs.
00608     Args:
00609         pollperiod: milliseconds, between 1 and 255; defaults to 50
00610
00611     Returns:
00612         tuple (error string, response string from setSubregister() OR invalid time
00613         setting string)
00614     """
00615     if pollperiod is None:
00616         pollperiod = 50
00617     if not isinstance(pollperiod, int) or pollperiod < 1 or pollperiod > 255:
00618         err = (
00619             self.logerr + "invalid poll period submitted. Setting remains "

```



```

00620         "unchanged. "
00621     )
00622     logging.error(err)
00623     return err, str(pollperiod)
00624 else:
00625     binset = bin(pollperiod)[2:].zfill(8)
00626     return self.ca.setSubregister("PPER", binset)
00627
00628 def getTemp(self, scale=None):
00629     """
00630     Read temperature sensor
00631     Args:
00632         scale: temperature scale to report (defaults to C, options are F and K)
00633
00634     Returns:
00635         temperature as float on given scale
00636     """
00637     err, rval = self.ca.getMonV("MON_TEMP", errflag=True)
00638     if err:
00639         logging.error(
00640             self.logerr + "unable to retrieve temperature information ("
00641             'getTemp), returning "0" '
00642         )
00643         return 0.0
00644     ctemp = rval * 1000 - 273.15
00645     if scale == "K":
00646         temp = ctemp + 273.15
00647     elif scale == "F":
00648         temp = 1.8 * ctemp + 32
00649     else:
00650         temp = ctemp
00651     return temp
00652
00653 def getPressure(self, offset=None, sensitivity=None, units=None):
00654     """
00655     Read pressure sensor. Uses default offset and sensitivity defined in init
00656     function unless alternatives are specified. NOTE: to reset defaults, reassign
00657     board.defoff and board.defsens explicitly
00658
00659     Args:
00660         offset: non-default offset in mv/V
00661         sensitivity: non-default sensitivity in mV/V/span
00662         units: units to report pressure (defaults to Torr, options are psi, bar,
00663             inHg, atm)
00664
00665     Returns:
00666         Pressure as float in chosen units, defaults to torr
00667     """
00668     if offset is None:
00669         offset = self.defoff
00670     if sensitivity is None:
00671         sensitivity = self.defsens
00672     if units is None:
00673         units = "torr"
00674     pplus = self.ca.getMonV("MON_PRES_PLUS")
00675     pminus = self.ca.getMonV("MON_PRES_MINUS")
00676     delta = 1000 * (pplus - pminus)
00677     ratio = sensitivity / 30 # nominal is 21/30
00678     psi = (delta - offset) / ratio
00679     if units.lower() == "psi":
00680         press = psi
00681     elif units.lower() == "bar":
00682         press = psi / 14.504
00683     elif units.lower() == "atm":
00684         press = psi / 14.695
00685     elif units.lower() == "inHg":
00686         press = psi * 2.036
00687     else:
00688         press = 51.715 * psi # default to Torr
00689
00690     return press
00691
00692 def clearStatus(self):
00693     """
00694     Check status registers to clear them
00695
00696     Returns:
00697         error string
00698     """
00699     err1, rval = self.ca.getRegister("STAT_REG_SRC")
00700     err2, rval = self.ca.getRegister("STAT_REG2_SRC")

```

```

00701         err = err1 + err2
00702     if err:
00703         logging.error(self.logerr + "clearStatus failed")
00704     return err
00705
00706 def checkStatus(self):
00707     """
00708     Check status register, convert to reverse-order bit stream (i.e., bit 0 is
00709         statusbits[0])
00710
00711     Returns:
00712         bit string (no '0b') in reversed order
00713     """
00714     err, rval = self.ca.getRegister("STAT_REG")
00715     rvalbits = bin(int(rval, 16))[2:].zfill(32)
00716     statusbits = rvalbits[::-1]
00717     return statusbits
00718
00719 def checkStatus2(self):
00720     """
00721     Check second status register, convert to reverse-order bit stream (i.e., bit 0
00722         is statusbits[0])
00723
00724     Returns: bit string (no '0b') in reversed order
00725     """
00726     err, rval = self.ca.getRegister("STAT_REG2")
00727     rvalbits = bin(int(rval, 16))[2:].zfill(6)
00728     statusbits = rvalbits[::-1]
00729     return statusbits
00730
00731 def reportStatus(self):
00732     """
00733     Check contents of status register, print relevant messages
00734     """
00735     statusbits = self.checkStatus()
00736     statusbits2 = self.checkStatus2()
00737
00738     logging.info(self.loginf + "Status report:")
00739     if int(statusbits[0]):
00740         print(self.loginf + "Sensor read complete")
00741     if int(statusbits[1]):
00742         print(self.loginf + "Coarse trigger detected")
00743     if int(statusbits[2]):
00744         print(self.loginf + "Fine trigger detected")
00745     if int(statusbits[5]):
00746         print(self.loginf + "Sensor readout in progress")
00747     if int(statusbits[6]):
00748         print(self.loginf + "Sensor readout complete")
00749     if int(statusbits[7]):
00750         print(self.loginf + "SRAM readout started")
00751     if int(statusbits[8]):
00752         print(self.loginf + "SRAM readout complete")
00753     if int(statusbits[9]):
00754         print(self.loginf + "High-speed timing configuration started")
00755     if int(statusbits[10]):
00756         print(self.loginf + "All ADCs configured")
00757     if int(statusbits[11]):
00758         print(self.loginf + "All DACs configured")
00759     if int(statusbits[13]):
00760         print(self.loginf + "Timer has reset")
00761     if int(statusbits[14]):
00762         print(self.loginf + "Camera is Armed")
00763     if int(statusbits[16]):
00764         print(self.loginf + "High-speed timing configuration complete")
00765     self.ca.sensor.reportStatusSensor(statusbits, statusbits2)
00766     temp = int(statusbits[23:16:-1], 2) * 3.3 * 1000 / 4096
00767     logging.info(
00768         self.loginf + "Temperature reading: " + "{0:1.2f}".format(temp) + " C"
00769     )
00770     press = int(statusbits[:23:-1], 2) * 3.3 * 1000 / 4096
00771     logging.info(
00772         self.loginf
00773         + "Pressure sensor reading: "
00774         + "{0:1.2f}".format(press)
00775         + " mV"
00776     )
00777     if int(statusbits2[0]):
00778         print(self.loginf + "FPA_IF_TO")
00779     if int(statusbits2[1]):
00780         print(self.loginf + "SRAM_RO_TO")
00781     if int(statusbits2[2]):

```

```

00782         print(self.logininfo + "PixelRd Timeout Error")
00783     if int(statusbits2[3]):
00784         print(self.logininfo + "UART_TX_TO_RST")
00785     if int(statusbits2[4]):
00786         print(self.logininfo + "UART_RX_TO_RST")
00787
00788     def reportEdgeDetects(self):
00789         """
00790         Report edge detects
00791         """
00792         err, rval = self.ca.getRegister("STAT_EDGE_DETECTS")
00793         # shift to left to fake missing edge detect
00794         edgebits = bin(int(rval, 16) << 1)[2:].zfill(32)
00795         # reverse to get order matching assignment
00796         bitsrev = edgebits[::-1]
00797         detdict = {}
00798         bitidx = 0
00799         for frame in range(4):
00800             for vert in ("TOP", "BOT"):
00801                 for edge in range(1, 3):
00802                     for hor in ("A", "B"):
00803                         detname = (
00804                             "W"
00805                             + str(frame)
00806                             + "_"
00807                             + vert
00808                             + "_"
00809                             + hor
00810                             + "_EDGE"
00811                             + str(edge)
00812                         )
00813                         detdict[detname] = bitsrev[bitidx]
00814                         bitidx += 1
00815         # remove faked detect
00816         del detdict["W0_TOP_A_EDGE1"]
00817         logging.info(self.logininfo + "Edge detect report:")
00818         for key, val in detdict.items():
00819             logging.info(self.logininfo + key + ": " + val)
00820
00821     def dumpStatus(self):
00822         """
00823         Create dictionary of status values, DAC settings, monitor values, and register
00824         values.
00825
00826         Returns:
00827             dictionary of system diagnostic values
00828         """
00829         statusbits = self.checkStatus()
00830         statusbits2 = self.checkStatus2()
00831
00832         temp = int(statusbits[23:16:-1], 2) * 3.3 * 1000 / 4096
00833         press = int(statusbits[:23:-1], 2) * 3.3 * 1000 / 4096
00834
00835         statDict = OrderedDict(
00836             {
00837                 "Temperature sensor reading (Deg C)": "{0:1.2f}".format(temp),
00838                 "Pressure reading (Torr)": str(round(self.ca.getPressure(), 3)),
00839                 "Pressure sensor reading (mV)": "{0:1.2f}".format(press),
00840                 "Sensor read complete": str(statusbits[0]),
00841                 "Coarse trigger detected": str(statusbits[1]),
00842                 "Fine trigger detected": str(statusbits[2]),
00843                 "Sensor readout in progress": str(statusbits[5]),
00844                 "Sensor readout complete": str(statusbits[6]),
00845                 "SRAM readout started": str(statusbits[7]),
00846                 "SRAM readout complete": str(statusbits[8]),
00847                 "High-speed timing configured": str(statusbits[9]),
00848                 "All ADCs configured": str(statusbits[10]),
00849                 "All DACs configured": str(statusbits[11]),
00850                 "Timer has reset": str(statusbits[13]),
00851                 "Camera is Armed": str(statusbits[14]),
00852                 "FPA_IF_TO": str(statusbits2[0]),
00853                 "SRAM_RO_TO": str(statusbits2[1]),
00854                 "PixelRd Timeout Error": str(statusbits2[2]),
00855                 "UART_TX_TO_RST": str(statusbits2[3]),
00856                 "UART_RX_TO_RST": str(statusbits2[4]),
00857                 "PDBIAS Unready": str(statusbits2[5]),
00858             }
00859         )
00860
00861         if self.ca.sensorname == "icarus" or self.ca.sensorname == "icarus2":
00862             senslabs = {

```

```

00863         3: "W3_Top_A_Edge1 detected",
00864         4: "W3_Top_B_Edge1 detected",
00865         12: "HST_All_W_En detected",
00866     }
00867 else:
00868     senslabs = {
00869         3: "RSLROWOUTA",
00870         4: "RSLROWOUTB",
00871         12: "RSLNALLWENB",
00872         15: "RSLNALLWENA",
00873         16: "Config HST is done",
00874     }
00875     sensDict = {senslabs[x]: str(statusbits[x]) for x in senslabs.keys()}
00876
00877     DACDict = OrderedDict()
00878     MonDict = OrderedDict()
00879     for entry in self.subreg_aliases:
00880         if self.subreg_aliases[entry][0] == "D":
00881             val = str(round(self.ca.getPotV(entry), 3))
00882             DACDict["DAC_" + entry] = val
00883         else:
00884             val = str(round(self.ca.getMonV(entry), 3))
00885             MonDict[entry] = val
00886
00887     regDict = OrderedDict()
00888     for key in self.registers.keys():
00889         # Load in all registers except for the read-clear status registers.
00890         if key == "STAT_REG_SRC" or key == "STAT_REG2_SRC":
00891             pass
00892         else:
00893             err, rval = self.ca.getRegister(key)
00894             regDict[key] = rval
00895
00896     dumpDict = OrderedDict()
00897     for x in [statDict, sensDict, MonDict, DACDict, regDict]:
00898         dumpDict.update(x)
00899     return dumpDict
00900
00901
00902 """
00903 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00904 LLNL-CODE-838080
00905
00906 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00907 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00908 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00909 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00910 be made under this license.
00911 """

```

7.15 C:/Users/hill35/git/camera_python/nsCamera/CameraAssembler.py File Reference

Classes

- class [nsCamera.CameraAssembler.CameraAssembler](#)

Namespaces

- namespace [nsCamera](#)
- namespace [nsCamera.CameraAssembler](#)

7.16 CameraAssembler.py

[Go to the documentation of this file.](#)

```

00001 # -*- coding: utf-8 -*-
00002 """
00003 CameraAssembler assembles the separate camera parts into a camera object. This object
00004 controls a combination of three components:
00005
00006 1. board : FPGA board -- LLNL_V1, LLNL_V4
00007 2. comms: communication interface -- GigE, RS422
00008 3. sensor : sensor type -- icarus, icarus2, daedalus
00009
00010 Author: Jeremy Martin Hill (jerhill@llnl.gov)
00011 Author: Matthew Dayton (dayton5@llnl.gov)
00012
00013 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00014 LLNL-CODE-838080
00015
00016 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00017 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00018 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00019 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00020 be made under this license.
00021
00022 Version: 2.1.2 (February 2025)
00023 """
00024
00025 from __future__ import absolute_import
00026
00027 import importlib
00028 import inspect
00029 import logging
00030 import os
00031 import platform
00032 import socket
00033 import sys
00034 import time
00035 import h5py
00036 from datetime import datetime
00037
00038 import numpy as np
00039
00040 from nsCamera.utils.misc import (
00041     bytes2str,
00042     checkCRC,
00043     flattenlist,
00044     generateFrames,
00045     getEnter,
00046     partition,
00047     plotFrames,
00048     saveTiffs,
00049     str2bytes,
00050     str2ndarray,
00051 )
00052
00053 from nsCamera.utils.Packet import Packet
00054
00055 # TODO: move to Sphinx documentation
00056 # TODO: add pytest and tox scripts
00057
00058
00059 class CameraAssembler:
00060     """
00061     Code to assemble correct code to manage FPGA, frame grabber, and sensor
00062
00063     Exposed methods:
00064         initialize() - initializes board registers and pots, sets up sensor
00065         reinitialize() - initialize board and sensors, restore last known timer settings
00066         reboot() - perform software reset of board and reinitialize
00067         getBoardInfo() - parses FPGA_NUM register to retrieve board description
00068         getRegister(regname) - retrieves contents of named register
00069         setRegister(regname, string) - sets named register to given value
00070         getSubregister(subregname) - return substring of register identified in board
00071             attribute 'subregname'
00072         setSubregister(subregname, valstring) - replace substring of register identified
00073             in board attribute 'subregname' with 'valstring'
00074         submitMessages(messages) - set registers or subregisters based on list of
00075             destination/payload tuples
00076         getPot(potname) - returns float (0 < value < 1) corresponding to integer stored

```

```

00077         in pot or monitor 'potname'
00078     setPot(potname, value) - 0 < value < 1; sets named pot to fixed-point number =
00079         'value' * (maximum pot value)
00080     getPotV(potname) - returns voltage setting of 'potname'
00081     setPotV(potname, voltage) - sets named pot to voltage
00082     getMonV(monname) - returns voltage read by monitor 'monname' (or monitor
00083         associated with given potname)
00084     readImgs() - calls arm() and readoff() functions
00085     saveFrames(frames) - save image object as one file
00086     saveNumpys(frames) - save individual frames as numpy data files
00087     dumpNumpy(datastream) - save datastream string to numpy file
00088     powerCheck(delta) - check that board power has not failed
00089     printBoardInfo() - print board information derived from FPGA_NUM register
00090     dumpRegisters() - return contents of all board registers
00091     dumpSubregisters() - return contents of all named board subregisters
00092     setFrames(min, max) - select subset of frames for readoff
00093     setRows(min, max, padToFull) - select subset of rows for readoff
00094     abortReadoff() - cancel readoff in wait-for-SRAM loop
00095     batchAcquire() - fast acquire a finite series of images
00096     loadTextFrames() - load data sets previously saved as text and convert to frames
00097
00098     Includes aliases to board- and sensor- specific methods:
00099     Board methods
00100         disarm() - take camera out of waiting-for-trigger state
00101         clearStatus() - clear contents of status registers
00102         checkStatus() - print contents of status register as reversed bit string
00103         checkStatus2() - print contents of status register 2 as reversed bit string
00104         reportStatus() - print report on contents of status registers
00105         resetTimer() - reset on-board timer
00106         getTimer() - read on-board timer
00107         enableLED(status) - enable (default) or disable (status = 0) on-board LEDs
00108         setLED(LED#, status) - turn LED on (default) or off (status = 0)
00109         setPowerSave(status) - turn powersave functionality on (default) or off
00110             (status = 0)
00111         getTemp() - read on-board temperature sensor
00112         getPressure() - read on-board pressure sensor
00113         dumpStatus() - generate dictionary of status, register, and subregister
00114             contents
00115     Sensor methods
00116         checkSensorVoltStat() - checks that jumper settings match sensor selection
00117         setTiming(side, sequencetuple, delay) - configure high-speed timing
00118         setArbTiming(side, sequencelist) - configure arbitrary high-speed timing
00119             sequence
00120         getTiming(side) - returns high speed timing settings from registers
00121         setManualTiming() - configures manual shutter timing
00122         getManualTiming() - returns manual shutter settings from registers
00123         selectOscillator(osc) - select timing oscillator
00124         setInterlacing(ifactor) - sets interlacing factor
00125         setHighFullWell(flag) - controls High Full Well mode
00126         setZeroDeadTime(flag, side) - controls Zero Dead Time mode
00127         setTriggerDelay(delayblocks) - sets trigger delay
00128     Comms methods
00129         sendCMD(pkt)- sends packet object via serial port
00130         arm() - configures software buffers & arms camera
00131         readFrames() - waits for data ready flag, then downloads image data
00132         readoff() - waits for data ready flag, then downloads image data
00133         closeDevice() - disconnect interface and release resources
00134     Miscellaneous functions (bare functions that can be called as methods)
00135         saveTiffs(frames) - save individual frames as tiffs
00136         plotFrames(frames) - plot individual frames as tiffs
00137
00138     Informational class variables:
00139         version - nsCamera software version
00140         FPGAVersion - firmware version (date)
00141         FPGAEnum - firmware implementation identifier
00142         FPGAboardtype - FPGA self-identified board type (should match 'boardname')
00143         FPGArad = Flag indicating radiation-tolerant FPGA build
00144         FPGAAsensor = FPGA self-identified sensor family (should correspond to
00145             'sensorname')
00146         FPGAinterfaces = FPGA self-identified interfaces (list should include
00147             'commname')
00148         FPGAinvalid = flag indicating invalid FPGA information in register
00149     """
00150
00151     def __init__(
00152         self,
00153         boardname="11n1_v4",
00154         commname="GigE",
00155         sensorname="icarus2",
00156         verbose=4,
00157         port=None,

```

```

00158         ip=None,
00159         logfile=None,
00160         logtag=None,
00161         timeout=30,
00162     ):
00163         """
00164         Args:
00165             boardname: name of FPGA board: llnl_v1, llnl_v4
00166             commname: name of communication interface: rs422, gige
00167             sensorname: name of sensor: icarus, icarus2, daedalus
00168             verbose: optional, sets logging level
00169                 0: print no logging messages
00170                 1: print CRITICAL logging messages (camera will not operate, e.g.,
00171                    unable to connect to board)
00172                 2: print ERROR logging messages (camera will not operate as directed,
00173                    e.g., an attempt to set the timing mode has failed, but the camera
00174                    is still operational)
00175                 3: print WARNING logging messages (camera will operate as directed, but
00176                    perhaps not as expected, e.g., ca.setTiming('A', (9, 8), 1) may be
00177                    programmed correctly, but the actual timing generated by the board
00178                    will be {1} [9, 8, 9, 14, 9, 8, 9])
00179                 4: print INFO logging messages (operational messages from ordinary
00180                    camera operation)
00181                 5: print DEBUG logging messages (detailed messages describing specific
00182                    operations and messages)
00183             port: optional integer
00184                 When using RS422, this preselects the comport for RS422 and bypasses
00185                 port search
00186                 When using GigE, this preselects the OrangeTree control port for GigE
00187                 (ignored if an ip parameter is not also provided)
00188             ip: optional string (e.g., '192.168.1.100')
00189                 GigE: bypasses network search and selects particular OrangeTree board -
00190                 required for some operating systems
00191             logfile: optional string, name of file to divert console output
00192             timeout: timeout in seconds for connecting using Gigabit Ethernet
00193         """
00194         self.version = "2.1.2"
00195         self.currttime = 0
00196         self.OLDtime = 0
00197         self.trigtime = []
00198         self.waited = []
00199         self.read = []
00200         self.unstringed = []
00201         self.parsedtime = []
00202         self.savetime = []
00203         self.cycle = []
00204         self.boardname = boardname.lower()
00205         self.timeout = timeout
00206         # TODO: parse boardname, etc. in separate method
00207         if self.boardname in ["llnlv1", "v1", "1", 1]:
00208             self.boardname = "llnl_v1"
00209         if self.boardname in ["llnlv4", "v4", "4", 4]:
00210             self.boardname = "llnl_v4"
00211         self.commname = commname.lower()
00212         if self.commname[0] == "g" or self.commname[0] == "e":
00213             self.commname = "gige"
00214         if self.commname[0] == "r":
00215             self.commname = "rs422"
00216         self.sensorname = sensorname.lower()
00217         if self.sensorname in ["i1", "ic1", "icarus1"]:
00218             self.sensorname = "icarus"
00219         if self.sensorname in ["i2", "ic2"]:
00220             self.sensorname = "icarus2"
00221         if self.sensorname == "d":
00222             self.sensorname = "daedalus"
00223         self.verbose = int(verbose)
00224         self.port = port
00225         self.python, self.pyth1, self.pyth2, _, _ = sys.version_info
00226         self.PY3 = self.python >= 3
00227         self.platform = platform.system()
00228         self.arch, _ = platform.architecture()
00229
00230         self.FPGAVersion = ""
00231         self.FPGANum = ""
00232         # FPGA information here and below populated during initialization using
00233         # getBoardInfo
00234         self.FPGAboardtype = ""
00235         self.FPGArad = False
00236         self.FPGAensor = ""
00237         self.FPGAinterfaces = []
00238

```

```

00239         # indicates invalid FPGA information in register# (0x80000001 accepted as valid)
00240         self.FPGAinvalid = False
00241
00242         self.iplist = None
00243         self.packageroot = os.path.dirname(inspect.getfile(CameraAssembler))
00244         self.armed = False
00245
00246         # only one of these collections (senstiming, sensmanual) should be nonempty at
00247         # any given time
00248         self.senstiming = {} # preserve HST setting against possible power failure
00249         self.sensmanual = [] # preserve manual timing
00250         self.inittime = 0
00251         self.padToFull = False
00252         self.abort = False
00253
00254         self.verbmap = {
00255             0: 99,
00256             1: logging.CRITICAL,
00257             2: logging.ERROR,
00258             3: logging.WARNING,
00259             4: logging.INFO,
00260             5: logging.DEBUG,
00261         }
00262         if logtag is None:
00263             logtag = ""
00264         self.logtag = logtag
00265
00266         self.logcritbase = "CRITICAL {logtag}: ".format(logtag=logtag)
00267         self.logerrbase = "ERROR {logtag}: ".format(logtag=logtag)
00268         self.logwarnbase = "WARNING {logtag}: ".format(logtag=logtag)
00269         self.loginfobase = "INFO {logtag}: ".format(logtag=logtag)
00270         self.logdebugbase = "DEBUG {logtag}: ".format(logtag=logtag)
00271
00272         self.logcrit = "{lb}[CA]".format(lb=self.logcritbase)
00273         self.logerr = "{lb}[CA]".format(lb=self.logerrbase)
00274         self.logwarn = "{lb}[CA]".format(lb=self.logwarnbase)
00275         self.loginfo = "{lb}[CA]".format(lb=self.loginfobase)
00276         self.logdebug = "{lb}[CA]".format(lb=self.logdebugbase)
00277
00278         self.verblevel = self.verbmap.get(verbose, 5) # defaults to 5 for invalid entry
00279
00280         if logfile:
00281             logging.basicConfig(format="%(message)s", filename=logfile)
00282         else:
00283             logging.basicConfig(format="%(message)s")
00284         logging.getLogger().setLevel(self.verblevel)
00285         logging.getLogger("matplotlib.font_manager").disabled = True
00286         logging.debug(
00287             "{logdebug}CameraAssembler: boardname = {boardname}; commname = {commname};"
00288             " sensorname = {sensorname}; verbose = {verbose}; port = {port}; ip = {ip};"
00289             " logfile = {logfile}; logtag = {logtag}".format(
00290                 logdebug=self.logdebug,
00291                 boardname=boardname,
00292                 commname=commname,
00293                 sensorname=sensorname,
00294                 verbose=verbose,
00295                 port=port,
00296                 ip=ip,
00297                 logfile=logfile,
00298                 logtag=logtag,
00299             )
00300         )
00301
00302         if ip:
00303             try:
00304                 iphex = socket.inet_aton(ip)
00305             except socket.error:
00306                 logging.critical(
00307                     "{logcrit}CameraAssembler: invalid IP provided".format(
00308                         logcrit=self.logcrit
00309                     )
00310                 )
00311                 sys.exit(1)
00312             ipnum = [0, 0, 0, 0]
00313             for i in range(4):
00314                 if self.PY3:
00315                     ipnum[i] = iphex[i]
00316                 else:
00317                     ipnum[i] = int(iphex[i].encode("hex"), 16)
00318             self.iplist = ipnum
00319

```



```
00320         self.payloaderror = False
00321
00322         # code pulled out of __init__ to facilitate reinitialization of the board
00323         # without needing to instantiate a new CameraAssembler object
00324         self.initialize()
00325
00326
00327
00328
00329     def initBoard(self):
00330         return self.board.initBoard()
00331
00332     def initPots(self):
00333         return self.board.initPots()
00334
00335     def latchPots(self):
00336         return self.board.latchPots()
00337
00338     def initSensor(self):
00339         return self.board.initSensor()
00340
00341     def configADCs(self):
00342         return self.board.configADCs()
00343
00344     def disarm(self):
00345         return self.board.disarm()
00346
00347     def startCapture(self, mode):
00348         return self.board.startCapture(mode)
00349
00350     def readSRAM(self):
00351         return self.board.readSRAM()
00352
00353     def waitforSRAM(self, timeout=None):
00354         return self.board.waitforSRAM(timeout)
00355
00356     def getTimer(self):
00357         return self.board.getTimer()
00358
00359     def resetTimer(self):
00360         return self.board.resetTimer()
00361
00362     def enableLED(self, status=1):
00363         return self.board.enableLED(status)
00364
00365     def setLED(self, LED=1, status=1):
00366         return self.board.setLED(LED, status)
00367
00368     def setPowerSave(self, status=1):
00369         return self.board.setPowerSave(status)
00370
00371     def setPPER(self, pollperiod=None):
00372         return self.board.setPPER(pollperiod)
00373
00374     def getTemp(self, scale=None):
00375         return self.board.getTemp(scale)
00376
00377     def getPressure(self, offset=None, sensitivity=None, units=None):
00378         return self.board.getPressure(offset, sensitivity, units)
00379
00380     def clearStatus(self):
00381         return self.board.clearStatus()
00382
00383     def checkStatus(self):
00384         return self.board.checkStatus()
00385
00386     def checkStatus2(self):
00387         return self.board.checkStatus2()
00388
00389     def reportStatus(self):
00390         return self.board.reportStatus()
00391
00392     def reportEdgeDetects(self):
00393         return self.board.reportEdgeDetects()
00394
00395     def dumpStatus(self):
00396         return self.board.dumpStatus()
00397
00398     def checkSensorVoltStat(self):
00399         return self.sensor.checkSensorVoltStat()
00400
00401     def setTiming(self, side="AB", sequence=None, delay=0):
```

```

00402         return self.sensor.setTiming(side, sequence, delay)
00403
00404     def setArbTiming(self, side="AB", sequence=None):
00405         return self.sensor.setArbTiming(side, sequence)
00406
00407     def getTiming(self, side=None, actual=None):
00408         return self.sensor.getTiming(side, actual)
00409
00410     def setManualShutters(self, timing=None):
00411         return self.sensor.setManualTiming(timing)
00412
00413     def setManualTiming(self, timing=None):
00414         return self.sensor.setManualTiming(timing)
00415
00416     def getManualTiming(self):
00417         return self.sensor.getManualTiming()
00418
00419     def getSensTemp(self, scale=None, offset=None, slope=None, dec=1):
00420         return self.sensor.getSensTemp(scale, offset, slope, dec)
00421
00422     def sensorSpecific(self):
00423         return self.sensor.sensorSpecific()
00424
00425     def selectOscillator(self, osc=None):
00426         return self.sensor.selectOscillator(osc)
00427
00428     def setInterlacing(self, ifactor=None, side=None):
00429         return self.sensor.setInterlacing(ifactor, side)
00430
00431     def setHighFullWell(self, flag=True):
00432         return self.sensor.setHighFullWell(flag)
00433
00434     def setZeroDeadTime(self, flag=True, side=None):
00435         return self.sensor.setZeroDeadTime(flag, side)
00436
00437     def setTriggerDelay(self, delay=0):
00438         return self.sensor.setTriggerDelay(delay)
00439
00440     def setPhiDelay(self, side=None, delay=0):
00441         return self.sensor.setPhiDelay(side, delay)
00442
00443     def setExtClk(self, dilation=None, frequency=None):
00444         return self.sensor.setExtClk(dilation, frequency)
00445
00446     def parseReadoff(self, frames, columns=1):
00447         return self.sensor.parseReadoff(frames, columns)
00448
00449     def sendCMD(self, pkt):
00450         return self.comms.sendCMD(pkt)
00451
00452     def arm(self, mode=None):
00453         return self.comms.arm(mode)
00454
00455     def readFrames(self, waitOnSRAM=None, timeout=0, fast=False, columns=1):
00456         frames, __, _ = self.comms.readoff(waitOnSRAM, timeout, fast, columns)
00457         return frames
00458
00459     def readoff(self, waitOnSRAM=None, timeout=0, fast=None, columns=1):
00460         return self.comms.readoff(waitOnSRAM, timeout, fast, columns)
00461
00462     def writeSerial(self, cmd, timeout=None):
00463         return self.comms.writeSerial(cmd, timeout)
00464
00465     def readSerial(self, size, timeout=None):
00466         return self.comms.readSerial(size, timeout)
00467
00468     def closeDevice(self):
00469         return self.comms.closeDevice()
00470
00471     def saveTiffs(self, frames, path=None, filename="Frame", prefix=None, index=None):
00472         return saveTiffs(self, frames, path, filename, prefix, index)
00473
00474     def plotFrames(self, frames, index=None):
00475         return plotFrames(self, frames, index)
00476
00477     def getEnter(self, text):
00478         return getEnter(text)
00479
00480     def checkCRC(self, rval):
00481         return checkCRC(rval)
00482

```

```

00483     def str2bytes(self, astring):
00484         return str2bytes(astring)
00485
00486     def bytes2str(self, bytessequence):
00487         return bytes2str(bytessequence)
00488
00489     def str2nparray(self, valstring):
00490         return str2nparray(valstring)
00491
00492     def flattenlist(self, mylist):
00493         return flattenlist(mylist)
00494
00495     def partition(self, frames, columns):
00496         return partition(self, frames, columns)
00497
00498
00499
00500     def initialize(self):
00501         """
00502         Initialize board registers and set pots
00503         """
00504         # TODO: automate sensor and board selection from firmware info
00505
00506
00507
00508         # get sensor
00509         # TODO: pull sensor, board, comm id out to separate methods
00510         if self.sensorname == "icarus":
00511             import nsCamera.sensors.icarus as snsr
00512         elif self.sensorname == "icarus2":
00513             import nsCamera.sensors.icarus2 as snsr
00514         elif self.sensorname == "daedalus":
00515             import nsCamera.sensors.daedalus as snsr
00516         else: # catch-all for added sensors to attempt object encapsulation
00517             sensormodname = ".sensors." + self.sensorname
00518             try:
00519                 sensormod = importlib.import_module(sensormodname, "nsCamera")
00520             except ImportError:
00521                 logging.critical(self.logcrit + "invalid sensor name")
00522                 sys.exit(1)
00523             snsr = getattr(sensormod, self.sensorname)
00524         self.sensor = snsr(self)
00525
00526         # kill existing connections (for reinitialize)
00527         if hasattr(self, "comms"):
00528             self.closeDevice()
00529
00530         # get communications interface
00531         if self.commname == "rs422":
00532             import nsCamera.comms.RS422 as comms
00533         elif self.commname == "gige":
00534             import nsCamera.comms.GigE as comms
00535         else:
00536             commsmodname = ".comms." + self.commname
00537             try:
00538                 commsmod = importlib.import_module(commsmodname, "nsCamera")
00539             except ImportError:
00540                 logging.critical(self.logcrit + "invalid comms name")
00541                 sys.exit(1)
00542             comms = getattr(commsmod, self.commname)
00543         self.comms = comms(self)
00544
00545         # get board
00546         if self.boardname == "llnl_v1":
00547             import nsCamera.boards.LLNL_v1 as brd
00548
00549             self.board = brd.llnl_v1(self)
00550         elif self.boardname == "llnl_v4":
00551             import nsCamera.boards.LLNL_v4 as brd
00552
00553             self.board = brd.llnl_v4(self)
00554         else:
00555             boardmodname = ".board." + self.boardname
00556             try:
00557                 boardmod = importlib.import_module(boardmodname, "nsCamera")
00558             except ImportError:
00559                 logging.critical(self.logcrit + "invalid board name")
00560                 sys.exit(1)
00561             boardobj = getattr(boardmod, self.boardname)
00562             self.board = boardobj(self)
00563
00564         # Now that board exists, initialize board-specific aliases for sensors

```

```

00565         self.sensor.init_board_specific()
00566
00567
00568
00569         # TODO: make cython the standard version
00570         # #####
00571         # # For cython version
00572         #
00573         # # get sensor
00574         # if self.sensorname == "icarus":
00575         #     import nsCamera.sensors.icarus as snsr
00576         #     self.sensor = snsr.icarus(self)
00577         # elif self.sensorname == "icarus2":
00578         #     import nsCamera.sensors.icarus2 as snsr
00579         #     self.sensor = snsr.icarus2(self)
00580         # elif self.sensorname == "daedalus":
00581         #     import nsCamera.sensors.daedalus as snsr
00582         #     self.sensor = snsr.daedalus(self)
00583         #
00584         # # kill existing connections (for reinitialize)
00585         # if hasattr(self, "comms"):
00586         #     self.closeDevice()
00587         #
00588         # # get communications interface
00589         # if self.commname == "rs422":
00590         #     import nsCamera.comms.RS422 as comms
00591         #     self.comms = comms.RS422(self)
00592         # elif self.commname == "gige":
00593         #     import nsCamera.comms.GigE as comms
00594         #     self.comms = comms.GigE(self)
00595         #
00596         # # get board
00597         # if self.boardname == "llnl_v1":
00598         #     import nsCamera.boards.LLNL_v1 as brd
00599         #     self.board = brd.llnl_v1(self)
00600         # elif self.boardname == "llnl_v4":
00601         #     import nsCamera.boards.LLNL_v4 as brd
00602         #     self.board = brd.llnl_v4(self)
00603         # #####
00604
00605         err, rval = self.getRegister("FPGA_NUM")
00606         if err or rval == "":
00607             err, rval = self.getRegister("FPGA_NUM")
00608             if err or rval == "":
00609                 logging.critical(
00610                     self.logcrit + "Initialization failed: unable to communicate with"
00611                     " board. "
00612                 )
00613                 sys.exit(1)
00614
00615         self.initBoard()
00616         self.initPots()
00617         self.initSensor()
00618         self.initPowerCheck()
00619         self.getBoardInfo()
00620         self.printBoardInfo()
00621
00622     def reinitialize(self):
00623         """
00624         Reinitialize board registers and pots, reinitialize sensor timing (if
00625             previously set)
00626         """
00627         logging.info(self.loginfo + "reinitializing")
00628         self.initialize()
00629
00630         for side in self.senstiming:
00631             self.setTiming(side, self.senstiming[side][0], self.senstiming[side][1])
00632
00633         if self.sensmanual: # should be mutually exclusive with anything in senstiming
00634             self.setManualShutters(self.sensmanual)
00635
00636     def reboot(self):
00637         """
00638         Perform soft reboot on board and reinitialize
00639         """
00640         self.board.softReboot()
00641         self.reinitialize()
00642
00643     def getBoardInfo(self):
00644         """
00645         Get board info from FPGA_NUM register. Returns error flag if register contents

```

```

00646         are invalid and tuple (board version number, rad tolerance flag, sensor name)
00647
00648     Returns:
00649         tuple (errorFlag, (board version, rad tolerance flag, sensor name))
00650     """
00651     invalidFPGANum = False
00652     interfaces = []
00653
00654     # TODO: move to new method (combine with parsing from initialize)
00655     if int(self.FPGANum[0], 16) & 8:
00656         if self.FPGANum[1] == "1":
00657             boardtype = "LLNLv1"
00658         elif self.FPGANum[1] == "4":
00659             boardtype = "LLNLv4"
00660         else:
00661             boardtype = "LLNLv?"
00662             invalidFPGANum = True
00663     else:
00664         boardtype = "SNLrevC"
00665         logging.warning(
00666             self.logwarn + "FPGA self-identifies as SNLrevC, which is not"
00667             " supported by this software "
00668         )
00669         invalidFPGANum = True
00670     self.FPGAboardtype = boardtype
00671
00672     if int(self.FPGANum[6], 16) & 1:
00673         rad = True
00674     else:
00675         rad = False
00676     self.FPGArad = rad
00677
00678     if self.FPGANum[7] == "1":
00679         sensor = "Icarus"
00680     elif self.FPGANum[7] == "2":
00681         sensor = "Daedalus"
00682     else:
00683         sensor = "Undefined"
00684         invalidFPGANum = True
00685     self.FPGAsensor = sensor
00686
00687     if int(self.FPGANum[5], 16) & 1:
00688         interfaces.append("RS422")
00689     if int(self.FPGANum[5], 16) & 2:
00690         interfaces.append("GigE")
00691     self.FPGAinterfaces = interfaces
00692
00693     if invalidFPGANum:
00694         if self.FPGANum == "80000001":
00695             invalidFPGANum = False
00696         else:
00697             logging.warning(self.logwarn + "FPGA self-identification is invalid")
00698     self.FPGAinvalid = invalidFPGANum
00699
00700     return invalidFPGANum, (boardtype, rad, sensor)
00701
00702 def getRegister(self, regname):
00703     """
00704     Retrieves contents of named register as hexadecimal string without '0x'
00705
00706     Args:
00707         regname: name of register as given in ICD
00708
00709     Returns:
00710         tuple: (error string, register contents as hexadecimal string without '0x')
00711     """
00712     # logging.debug(self.logdebug + "getRegister: regname = " + str(regname))
00713     logging.debug(
00714         "{logdebug}getRegister: regname = {regname}".format(
00715             logdebug=self.logdebug, regname=regname
00716         )
00717     )
00718
00719     regname = regname.upper()
00720     if regname not in self.board.registers:
00721         err = "{logerr}getRegister: Invalid register name: {regname}; returning"
00722         " zeros".format(logerr=self.logerr, regname=regname)
00723         logging.error(err)
00724         return err, "00000000"
00725     sendpkt = Packet(cmd="1", addr=self.board.registers[regname])
00726     err, rval = self.comms.sendCMD(sendpkt)

```

```

00727         if err:
00728             logging.error(
00729                 "{logerr}getRegister: {regname}; {err}".format(
00730                     logerr=self.logerr, regname=regname, err=err
00731                 )
00732             )
00733
00734         retval = rval[8:16]
00735         logging.debug(
00736             "{logdebug}getRegister: retval = {retval}".format(
00737                 logdebug=self.logdebug, retval=retval
00738             )
00739         )
00740
00741         return err, retval
00742
00743     def setRegister(self, regname, regval):
00744         """
00745         Sets named register to given value
00746
00747         Args:
00748             regname: name of register as given in ICD
00749             regval: value to assign to register, as integer or hexadecimal string
00750                     with or without '0x'
00751
00752         Returns:
00753             tuple: (error string, response string)
00754         """
00755         logging.debug(
00756             "{logdebug}setRegister: regname = {regname}; regval = {regval}".format(
00757                 logdebug=self.logdebug, regname=regname, regval=regval
00758             )
00759         )
00760
00761         regname = regname.upper()
00762         if regname not in self.board.registers:
00763             err = "{logerr}setRegister: Invalid register name: {regname}".format(
00764                 logerr=self.logerr, regname=regname
00765             )
00766             logging.error(err)
00767             return err, "00000000"
00768         if isinstance(regval, int):
00769             regval = hex(regval)
00770         try:
00771             if regval[0:2] == "0x":
00772                 regval = regval[2:]
00773         except TypeError:
00774             err = "{logerr}setRegister: invalid register value parameter".format(
00775                 logerr=self.logerr
00776             )
00777             logging.error(err)
00778             return err, "00000000"
00779         pkt = Packet(addr=self.board.registers[regname], data=regval)
00780         err, rval = self.comms.sendCMD(pkt)
00781         if err:
00782             logging.error(
00783                 "{logerr}setRegister: {regname}: {err}".format(
00784                     logerr=self.logerr, regname=regname, err=err
00785                 )
00786             )
00787         if len(rval) < 32:
00788             logging.debug(
00789                 "{logdebug}SetRegister: rval = {rval}".format(
00790                     logdebug=self.logdebug, rval=rval
00791                 )
00792             )
00793         else:
00794             logging.debug(
00795                 "{logdebug}SetRegister: rval (truncated)= {rval}".format(
00796                     logdebug=self.logdebug, rval=rval[0:32]
00797                 )
00798             )
00799         return err, rval
00800
00801     def resolveSubreg(self, srname):
00802         """
00803         Resolves subregister name or alias, returns object associated with subregister
00804         and flag indicating writability
00805
00806         Args:
00807             srname: name or alias of subregister

```

```

00808
00809     Returns:
00810         tuple(subregister name string, associated object, writable flag)
00811     """
00812     logging.debug(
00813         "{logdebug}resolveSubreg: srname = {srname}".format(
00814             logdebug=self.logdebug,
00815             srname=srname,
00816         )
00817     )
00818     writable = False
00819     srname = srname.upper()
00820     if srname in self.board.subreg_aliases:
00821         srname = self.board.subreg_aliases[srname].upper()
00822     if srname in self.board.subreglist:
00823         srobj = getattr(self.board, srname)
00824         writable = getattr(self.board, srname).writable
00825     else:
00826         # No-object error is handled by calling function
00827         srobj = None
00828     logging.debug(
00829         "{logdebug}resolveSubreg: srobj = {srobj}, writable={writable}".format(
00830             logdebug=self.logdebug, srobj=srobj, writable=writable
00831         )
00832     )
00833
00834     return srname, srobj, writable
00835
00836 def getSubregister(self, subregname):
00837     """
00838     Returns substring of register identified in board attribute 'subregname'
00839
00840     Args:
00841         subregname: listed in board.subreg_aliases or defined in board.subregisters
00842
00843     Returns:
00844         tuple: (error string, contents of subregister as binary string without '0b')
00845     """
00846     logging.debug(
00847         "{logdebug}getSubregister: subregname = {subregname}".format(
00848             logdebug=self.logdebug,
00849             subregname=subregname,
00850         )
00851     )
00852
00853     subregname, subregobj, _ = self.resolveSubreg(subregname)
00854     if not subregobj:
00855         err = "{logerr}getSubregister: invalid lookup: {subregname}; returning"
00856         err = err.format(logerr=self.logerr, subregname=subregname)
00857         logging.error(err)
00858         return err, "".zfill(8)
00859     err, resp = self.getRegister(subregobj.register)
00860     if err:
00861         logging.error(
00862             "{logerr}getSubregister: unable to retrieve register setting: \
00863             {subregname}; returning '0' string".format(
00864                 logerr=self.logerr, subregname=subregname
00865             )
00866         )
00867
00868     return err, "".zfill(8)
00869
00870     hex_str = "0x" + resp # this should be a hexadecimalstring
00871     b_reg_value = "{0:0=32b}".format(int(hex_str, 16)) # convert to binary string
00872     # list indexing is reversed from bit string; the last bit of the string is at
00873     # index 0 in the list (thus bit 0 is at index 0)
00874     startindex = 31 - subregobj.start_bit
00875     retval = b_reg_value[startindex : startindex + subregobj.width]
00876     logging.debug(
00877         "{logdebug}getSubregister: retval = {retval}".format(
00878             logdebug=self.logdebug, retval=retval
00879         )
00880     )
00881     return "", retval
00882
00883 def setSubregister(self, subregname, valstring):
00884     """
00885     Sets substring of register identified in board attribute 'subregname' to
00886     valstring if subregister is writable
00887
00888     Args:

```

```

00889         subregname: listed in board.subreg_aliases or defined in board.subregisters
00890         valstring: integer or binary string with or without '0b'
00891
00892     Returns:
00893         tuple: (error, packet response string) from setRegister
00894     """
00895     logging.debug(
00896         "{logdebug}setSubregister: subregname = {subregname}; valstring ="
00897         " {valstring}".format(
00898             logdebug=self.logdebug, subregname=subregname, valstring=valstring
00899         )
00900     )
00901
00902     subregname, subregobj, writable = self.resolveSubreg(subregname)
00903     if not subregobj:
00904         err = "{logerr}getSubregister: invalid lookup: {subregname}".format(
00905             logerr=self.logerr, subregname=subregname
00906         )
00907
00908         logging.error(err)
00909         return err, "0"
00910     if not writable:
00911         err = "{logerr}getSubregister: not a writable subregister: {subregname}"
00912         ".format(logerr=self.logerr, subregname=subregname)
00913         logging.error(err)
00914         return err, "0"
00915     if isinstance(valstring, int):
00916         valstring = bin(valstring)[2:]
00917     try:
00918         if valstring[0:2] == "0b":
00919             valstring = valstring[2:]
00920     except TypeError:
00921         err = "{logerr}getSubregister: invalid subregister value parameter".format(
00922             logerr=self.logerr
00923         )
00924
00925         logging.error(err)
00926         return err, "0"
00927     if len(str(valstring)) > subregobj.width:
00928         err = "{logerr}getSubregister: ivalue string is too long".format(
00929             logerr=self.logerr
00930         )
00931
00932         logging.error(err)
00933         return err, "0"
00934     # read current value of register data
00935     err, resp = self.getRegister(subregobj.register)
00936     if err:
00937         logging.error(
00938             "{logerr}getSubregister: unable to retrieve register setting; setting"
00939             " of {subregname} likely failed ".format(
00940                 logerr=self.logerr, subregname=subregname
00941             )
00942         )
00943
00944         return err, "0"
00945     hex_str = "0x" + resp
00946     b_reg_value = "{0:0=32b}".format(int(hex_str, 16)) # convert to binary
00947     # list indexing is reversed from bit string; the last bit of the string is at
00948     # index 0 in the list (thus bit 0 is at index 0)
00949     startindex = 31 - subregobj.start_bit
00950     valstringpadded = str(valstring).zfill(subregobj.width)
00951     fullreg = list(b_reg_value)
00952     fullreg[startindex : startindex + subregobj.width] = valstringpadded
00953     # convert binary string back to hexadecimal string for writing
00954     new_reg_value = "".join(fullreg)
00955     h_reg_value = "{num:{fill}{width}x}".format(
00956         num=int(new_reg_value, 2), fill="0", width=8
00957     )
00958     err, retval = self.setRegister(subregobj.register, h_reg_value)
00959     # logging.debug(self.logdebug + "retval = " + str(retval))
00960     if len(retval) < 32:
00961         logging.debug(
00962             "{logdebug}setSubregister: retval = {retval}".format(
00963                 logdebug=self.logdebug, retval=retval
00964             )
00965         )
00966     else:
00967         logging.debug(
00968             "{logdebug}setSubregister: retval (truncated) = {retval}".format(
00969                 logdebug=self.logdebug, retval=retval[0:32]

```



```

00970         )
00971     )
00972
00973     return err, retval
00974
00975 def submitMessages(self, messages, errorstring="Error"):
00976     """
00977     Serially set multiple register / subregister values
00978
00979     Args:
00980         messages: list of tuples (register name, integer or hexadecimal string with
00981             or without '0x') and/or (subregister name, integer or binary string with
00982             or without '0b')
00983         errorstring: error message to print in case of failure
00984
00985     Returns:
00986         tuple (accumulated error string, response string of final message)
00987     """
00988     logging.debug(
00989         "{logdebug}submitMessages: messages = {messages}; errorstring = "
00990         " {errorstring}".format(
00991             logdebug=self.logdebug, messages=messages, errorstring=errorstring
00992         )
00993     )
00994
00995     errs = ""
00996     err = ""
00997     rval = ""
00998     for m in messages:
00999         if m[0].upper() in self.board.registers:
01000             err, rval = self.setRegister(m[0].upper(), m[1])
01001         elif m[0].upper() in self.board.subreglist:
01002             err, rval = self.setSubregister(m[0].upper(), m[1])
01003         else:
01004             err = "{logerr}submitMessages: Invalid register/subregister:"
01005             " {errorstring}:{m0}; ".format(
01006                 logerr=self.logerr, errorstring=errorstring, m0=m[0]
01007             )
01008
01009             logging.error(err)
01010             errs = errs + err
01011     return err, rval
01012
01013 def getPot(self, potname, errflag=False):
01014     """
01015     Retrieves value of pot or ADC monitor subregister, scaled to [0,1).
01016
01017     Args:
01018         potname: name of pot or monitor, e.g., VRST or MON_CH2 found in
01019             board.subreg_aliases or defined in board.subregisters
01020         errflag: if True, return tuple with error string
01021
01022     Returns:
01023         if errflag:
01024             tuple: (error string, float value of subregister, scaled to [0,1) )
01025         else:
01026             float value of subregister, scaled to [0,1)
01027     """
01028     logging.debug(
01029         "{logdebug}getPot: potname = {potname}; errflag = {errflag}".format(
01030             logdebug=self.logdebug, potname=potname, errflag=errflag
01031         )
01032     )
01033
01034     potname, potobj, _ = self.resolveSubreg(potname)
01035     if not potobj:
01036         err = "{logerr}getPot: invalid lookup: {potname}; returning 0".format(
01037             logerr=self.logerr, potname=potname
01038         )
01039
01040         logging.error(err)
01041         if errflag:
01042             return err, 0
01043         return 0
01044     err, b_pot_value = self.getSubregister(potname)
01045     if err:
01046         err = "{logerr}getPot: unable to read subregister: {potname}".format(
01047             logerr=self.logerr, potname=potname
01048         )
01049
01050         # convert binary string back to decimal

```

```

01051         f_reg_value = 1.0 * int(b_pot_value, 2)
01052         value = (f_reg_value - potobj.min) / (potobj.max - potobj.min)
01053         # logging.debug(self.logdebug + "getpot: value = " + str(value))
01054
01055         logging.debug(
01056             "{logdebug}getpot: value = {value}".format(
01057                 logdebug=self.logdebug, value=value
01058             )
01059         )
01060
01061         if errflag:
01062             return err, value
01063         return value
01064
01065     def setPot(self, potname, value=1.0, errflag=False):
01066         """
01067         Sets value of pot to value, normalized so that '1.0' corresponds with the
01068             fixed point maximum value of pot.
01069
01070         Args:
01071             potname: common name of pot, e.g., VRST found in board.subreg_aliases or
01072                 defined in board.subregisters
01073             value: float between 0 and 1
01074             errflag: if True, return tuple with error string
01075
01076         Returns:
01077             if errflag:
01078                 tuple: (error string, response packet as string)
01079             else:
01080                 response packet as string
01081         """
01082         logging.debug(
01083             "{logdebug}setPot: potname = {potname}; value={value} errflag = {errflag}"
01084             "".format(
01085                 logdebug=self.logdebug, potname=potname, value=value, errflag=errflag
01086             )
01087         )
01088
01089         if value < 0:
01090             value = 0.0
01091         if value > 1:
01092             value = 1.0
01093
01094         potname, potobj, writable = self.resolveSubreg(potname)
01095         if not potobj:
01096             err = "{logerr}setPot: invalid lookup: {potname}; returning '0'".format(
01097                 logerr=self.logerr, potname=potname
01098             )
01099
01100             logging.error(err)
01101             if errflag:
01102                 return err, 0
01103             return 0
01104         if not writable:
01105             err = "{logerr}setPot: not a writable subregister: {potname}; returning '0'"
01106             "".format(logerr=self.logerr, potname=potname)
01107             logging.error(err)
01108             if errflag:
01109                 return err, "0"
01110             return "0"
01111         setpoint = int(round(value * potobj.max_value))
01112         setpointpadded = "{num:{fill}{width}b}".format(
01113             num=setpoint, fill="0", width=potobj.width
01114         )
01115         logging.debug(
01116             "{logdebug}setpot: setpointpadded = {setpointpadded}".format(
01117                 logdebug=self.logdebug, setpointpadded=setpointpadded
01118             )
01119         )
01120
01121         err, rval = self.setSubregister(potname, setpointpadded)
01122         if err:
01123             logging.error(
01124                 err="{logerr}setPot: unable to confirm setting of subregister:"
01125                 " {potname}".format(logerr=self.logerr, potname=potname)
01126             )
01127         ident = potname[3:]
01128         if ident[0].isdigit(): # numbered pot scheme
01129             potnumlatch = int(ident) * 2 + 1
01130             potnumlatchstring = "{num:{fill}{width}x}".format(
01131                 num=potnumlatch, fill="0", width=8

```

```

01132         )
01133         err1, resp = self.setRegister("POT_CTL", potnumlatchstring)
01134     else: # alphabetical DAC scheme
01135         ident = ident.upper() # expects single character, e.g. 'A' from 'DACA'
01136         identnum = ord(ident) - ord("A") # DACA -> 0
01137         potnumlatch = int(identnum) * 2 + 1
01138         potnumlatchstring = "{num:{fill}{width}x}".format(
01139             num=potnumlatch, fill="0", width=8
01140         )
01141         err1, resp = self.setRegister("DAC_CTL", potnumlatchstring)
01142     if err1:
01143         # logging.error(self.logerr + "setPot: unable to latch register")
01144
01145         logging.error(
01146             err="{logerr}setPot: unable to latch register".format(
01147                 logerr=self.logerr
01148             )
01149         )
01150
01151     if errflag:
01152         return err + err1, rval
01153     return rval
01154
01155 def getPotV(self, potname, errflag=False):
01156     """
01157     Reads voltage _setting_ (not actual voltage) of specified pot
01158
01159     Args:
01160         potname: name of pot or monitor, e.g., VRST or MON_CH2 found in
01161             board.subreg_aliases or defined in board.subregisters
01162         errflag: if True, return tuple with error string
01163
01164     Returns:
01165         if errflag:
01166             tuple: (error string, float value of pot voltage)
01167         else:
01168             float value of pot voltage
01169     """
01170     logging.debug(
01171         self.logdebug
01172         + "getPotV: potname = "
01173         + str(potname)
01174         + "; errflag = "
01175         + str(errflag)
01176     )
01177     potname, potobj, _ = self.resolveSubreg(potname)
01178     if not potobj:
01179         err = (
01180             self.logerr + "getPotV: invalid lookup: " + potname + " , returning 0 "
01181         )
01182         logging.error(err)
01183         if errflag:
01184             return err, 0
01185         return 0
01186     err, val = self.getPot(potname, errflag=True)
01187     logging.debug(self.logdebug + "getPotV: val = " + str(val))
01188     if err:
01189         logging.error(self.logerr + "getPotV: unable to read pot " + potname)
01190     minV = potobj.minV
01191     maxV = potobj.maxV
01192     if errflag:
01193         return err, val * (maxV - minV)
01194     return val * (maxV - minV)
01195
01196 # TODO: optimize tuning speed for DACs
01197 def setPotV(
01198     self,
01199     potname,
01200     voltage,
01201     tune=False,
01202     accuracy=0.01,
01203     iterations=20,
01204     approach=0.75,
01205     errflag=False,
01206 ):
01207     """
01208     Sets pot to specified voltage. If tune=True, uses monitor to adjust pot to
01209     correct voltage. Tuning will attempt to tune to closest LSB on pot; if
01210     'accuracy' > LSB resolution, will only complain if tuning is unable to get
01211     the voltage within 'accuracy'
01212 
```

```

01213     Args:
01214         potname: common name of pot, e.g., VRST found in board.subreg_aliases or
01215                 defined in board.subregisters
01216         voltage: voltage bound by pot max and min (set in board object)
01217         tune: if True, iterate with monitor to correct voltage
01218         accuracy: acceptable error in volts (if None, attempts to find the closest
01219                 possible pot setting and warns if last iteration does not reduce error
01220                 below the resolution of the pot)
01221         iterations: number of iteration attempts
01222         approach: approximation parameter (>1 may cause overshoot)
01223         errflag: if True, return tuple with error string
01224
01225     Returns:
01226         if errflag:
01227             tuple: (error string, response string)
01228         else:
01229             response string
01230
01231     """
01232     logging.debug(
01233         self.logdebug
01234         + "setPotV: potname = "
01235         + str(potname)
01236         + "; voltage = "
01237         + str(voltage)
01238         + "; tune = "
01239         + str(tune)
01240         + "; accuracy = "
01241         + str(accuracy)
01242         + "; iterations = "
01243         + str(iterations)
01244         + "; approach = "
01245         + str(approach)
01246         + "; errflag = "
01247         + str(errflag)
01248     )
01249     potname, potobj, writable = self.resolveSubreg(potname)
01250     if not potobj:
01251         err = (
01252             self.logerr
01253             + "setPotV: invalid lookup: "
01254             + potname
01255             + " , returning zero "
01256         )
01257         logging.error(err)
01258         if errflag:
01259             return err, 0
01260         return 0
01261     if not writable:
01262         err = (
01263             self.logerr
01264             + "setPotV: not a writable subregister: "
01265             + potname
01266             + "; returning zero"
01267         )
01268         logging.error(err)
01269         if errflag:
01270             return err, 0
01271         return 0
01272     if voltage < potobj.minV:
01273         voltage = potobj.minV
01274     if voltage > potobj.maxV:
01275         voltage = potobj.maxV
01276     setting = (voltage - potobj.minV) / (potobj.maxV - potobj.minV)
01277     logging.debug(self.logdebug + "setPotV: setting = " + str(setting))
01278     err, rval = self.setPot(potname, setting, errflag=True)
01279     time.sleep(0.1)
01280     # TODO: refactor tuning to separate method
01281     if tune:
01282         logging.debug(self.logdebug + "setPotV: beginning tuning")
01283         if potname not in self.board.monitor_controls.values():
01284             err = (
01285                 self.logerr
01286                 + "setPotV: pot ' "
01287                 + potname
01288                 + "' does not have a corresponding monitor"
01289             )
01290             logging.error(err)
01291             if errflag:
01292                 return err, rval
01293             return rval
01294     self.setPot(potname, 0.65)

```

```

01294         time.sleep(0.2)
01295         err1, mon65 = self.getMonV(potname, errflag=True)
01296         self.setPot(potname, 0.35)
01297         time.sleep(0.2)
01298         err2, mon35 = self.getMonV(potname, errflag=True)
01299         # theoretical voltage range assuming linearity
01300         potrange = (mon65 - mon35) / 0.3
01301         stepsize = potrange / (potobj.max_value + 1)
01302         err += err1 + err2
01303         if err or potrange < 1:
01304             err += self.logerr + "setPotV: unable to tune pot " + potname
01305             if potrange < 1: # potrange should be on the order of 3.3 or 5 volts
01306                 err += "; monitor shows insufficient change with pot variation; "
01307                 "retrying setPotV with tune=False"
01308             logging.warning(err)
01309             err, rval = self.setPotV(
01310                 potname=potname, voltage=voltage, tune=False, errflag=True
01311             )
01312             if errflag:
01313                 return err, rval
01314             return rval
01315         potzero = 0.35 - (mon35 / potrange)
01316         potone = 1.65 - (mon65 / potrange)
01317         if potzero < 0:
01318             potzero = 0
01319         if potone > 1:
01320             potone = 1
01321
01322         if accuracy > stepsize:
01323             mindiff = accuracy
01324         else:
01325             mindiff = stepsize
01326         setting = potzero + (voltage / potone)
01327         self.setPot(potname, setting)
01328         lastdiff = 0
01329         smalladjust = 0
01330         err3 = ""
01331         for _ in range(iterations):
01332             err3i, measured = self.getMonV(potname, errflag=True)
01333             if err3i:
01334                 err3 = err3 + err3i + " "
01335             diff = voltage - measured
01336             if abs(diff - lastdiff) < stepsize / 2:
01337                 if (
01338                     smalladjust > 12
01339                 ): # magic number for now; if it doesn't converge after several
01340                     # tries, it never will, usually because the setting is pinned
01341                     # to 0 or 1 and adjust can't change it
01342                     logging.warning(
01343                         self.logwarn
01344                         + "setPotV: Tuning converged too slowly: pot "
01345                         + potname
01346                         + " set to "
01347                         + str(voltage)
01348                         + "V, monitor returns "
01349                         + str(measured)
01350                         + "V; if this value is incorrect, consider trying "
01351                         + "tune=False"
01352                     )
01353                     logging.debug(self.logdebug + "setPotV: tuning complete")
01354                     if errflag:
01355                         return "", rval
01356                     return rval
01357                 smalladjust += 1
01358             if not int(2 * diff / stepsize):
01359                 # TODO: is this check redundant with the first one?
01360                 logging.debug(self.logdebug + "setPotV: tuning complete")
01361                 if errflag:
01362                     return "", rval
01363                 return rval
01364             adjust = approach * (diff / potrange)
01365             setting += adjust
01366             if setting > 1:
01367                 setting = 1
01368             elif setting < 0:
01369                 setting = 0
01370             err1, rval = self.setPot(potname, setting, True)
01371             lastdiff = diff
01372             time.sleep(0.2)
01373             err4, measured = self.getMonV(potname, errflag=True)
01374             diff = voltage - measured

```

```

01375         # code will try to get to within one stepsize, but will only complain if it
01376         # doesn't get within mindiff
01377         if int(diff / mindiff):
01378             logging.warning(
01379                 self.logwarn
01380                 + "setPotV: pot "
01381                 + potname
01382                 + " set to "
01383                 + str(voltage)
01384                 + "V, monitor returns "
01385                 + str(measured)
01386                 + "V"
01387             )
01388             err += err1 + err2 + err3 + err4
01389         if err:
01390             logging.error(self.logerr + "setPotV: errors occurred: " + err)
01391         if errflag:
01392             return err, rval
01393         logging.debug(self.logdebug + "setPotV: tuning complete")
01394         return rval
01395
01396     def getMonV(self, monname, errflag=False):
01397         """
01398         Reads voltage from monitor named or that associated with the pot named 'monname'
01399
01400         Args:
01401             monname: name of pot or monitor, e.g., VRST or MON_CH2 found in
01402                     board.subreg_aliases or defined in board.subregisters
01403             errflag: if True, return tuple with error string
01404
01405         Returns:
01406             if errflag:
01407                 tuple: (error string, float value of voltage measured by monitor)
01408             else:
01409                 float value of voltage measured by monitor
01410         """
01411         logging.debug(
01412             self.logdebug
01413             + "getMonV: monname = "
01414             + str(monname)
01415             + "; errflag = "
01416             + str(errflag)
01417         )
01418         monname = monname.upper()
01419         if monname in self.board.subreg_aliases:
01420             monname = self.board.subreg_aliases[monname].upper()
01421         # else:
01422         for key, value in self.board.monitor_controls.items():
01423             if value == monname:
01424                 monname = key
01425         if monname not in self.board.monitor_controls:
01426             if monname in self.board.subreglist:
01427                 pass # no change necessary
01428             else:
01429                 err = (
01430                     self.logerr + "getMonV: invalid lookup " + monname + ", returning 0"
01431                 )
01432                 logging.error(err)
01433                 if errflag:
01434                     return err, 0
01435                 return 0
01436         err, monval = self.getPot(monname, errflag=True)
01437         logging.debug(self.logdebug + "getMonV: monval = " + str(monval))
01438         if err:
01439             logging.error(
01440                 self.logerr + "getMonV: unable to read monitor value for " + monname
01441             )
01442         if self.board.ADC5_bipolar:
01443             if monval >= 0.5:
01444                 monval -= 1 # handle negative measurements (two's complement)
01445             if errflag:
01446                 return err, 2 * self.board.ADC5_mult * monval * self.board.VREF
01447             return 2 * self.board.ADC5_mult * monval * self.board.VREF
01448         else:
01449             if errflag:
01450                 return err, self.board.ADC5_mult * monval * self.board.VREF
01451             return self.board.ADC5_mult * monval * self.board.VREF
01452
01453     def readImgs(self, waitOnSRAM=True, mode="Hardware"):
01454         """
01455         Combines arm() and readoff() functions

```

```

01456
01457     Returns:
01458         tuple (list of numpy arrays, length of downloaded payload, payload error
01459             flag) returned by readoff
01460     """
01461     logging.info(self.logininfo + "readImgs")
01462     self.arm(mode)
01463     return self.readoff(waitOnSRAM)
01464
01465 def saveFrames(self, frames, path=None, filename="frames", prefix=None):
01466     """
01467     Save list of numpy arrays to disk. If passed an unprocessed text string, saves
01468     it directly to disk for postprocessing. Use 'prefix=""' for no prefix
01469
01470     Args:
01471         frames: numpy array or list of numpy arrays OR text string
01472         path: save path, defaults to './output'
01473         filename: defaults to 'frames.bin'
01474         prefix: prepended to filename, defaults to time/date (e.g. '160830-124704_')
01475
01476     Returns:
01477         Error string
01478     """
01479     logging.debug(
01480         self.logdebug
01481         + "saveFrames: path = "
01482         + str(path)
01483         + "; filename = "
01484         + str(filename)
01485         + "; prefix = "
01486         + str(prefix)
01487     )
01488     logging.info(self.logininfo + "saveFrames")
01489     err = ""
01490     if path is None:
01491         path = os.path.join(os.getcwd(), "output")
01492     if prefix is None:
01493         prefix = datetime.now().strftime("%y%m%d-%H%M%S%f")[:-5] + "_"
01494     if not os.path.exists(path):
01495         os.makedirs(path)
01496
01497     # TODO catch save file exceptions
01498     if isinstance(frames[0], str):
01499         logging.debug(self.logdebug + "saveFrames: saving text frames")
01500         filename = filename + ".txt"
01501         savefile = open(os.path.join(path, prefix + filename), "w+")
01502         savefile.write(frames)
01503     else:
01504         logging.debug(self.logdebug + "saveFrames: saving numerical frames")
01505         filename = filename + ".bin"
01506         stacked = np.stack(frames)
01507         try:
01508             stacked = stacked.reshape(
01509                 (
01510                     self.sensor.nframes,
01511                     self.sensor.height // (self.sensor.interlacing + 1),
01512                     self.sensor.width,
01513                 )
01514             )
01515         except Exception as e:
01516             err = self.logerr + "saveFrames: unable to save frames: " + str(e)
01517             logging.error(err)
01518
01519         stacked.tofile(os.path.join(path, prefix + filename))
01520     return err
01521
01522 def saveNumpys(
01523     self,
01524     frames,
01525     path=None,
01526     filename="Frame",
01527     prefix=None,
01528     index=None,
01529 ):
01530     """
01531     Save numpy array or list of numpy arrays to disk as individual numpy data files,
01532     with frame number appended to filename.
01533
01534     Args:
01535         frames: numpy array or list of numpy arrays or single numpy array
01536         path: save path, defaults to './output'

```

```

01537         filename: defaults to 'Frame' followed by frame number
01538         prefix: prepended to 'filename', defaults to time/date
01539             (e.g. '160830-124704_')
01540         index: number to start frame numbering
01541
01542     Returns:
01543         Error string
01544     """
01545     logging.info(self.loggerinfo + "saveNumpys")
01546     logging.debug(
01547         self.loggerdebug
01548         + "saveNumpys: path = "
01549         + str(path)
01550         + "; filename = "
01551         + str(filename)
01552         + "; prefix = "
01553         + str(prefix)
01554         + "; index = "
01555         + str(index)
01556     )
01557     err = ""
01558     if path is None:
01559         path = os.path.join(os.getcwd(), "output")
01560     if prefix is None:
01561         prefix = datetime.now().strftime("%Y%m%d-%H%M%S%f")[:-5] + "_"
01562     if not os.path.exists(path):
01563         os.makedirs(path)
01564     if index is None:
01565         firstnum = self.sensor.firstframe
01566     else:
01567         firstnum = index
01568     if not isinstance(frames, list):
01569         frames = [frames]
01570
01571     # if this is a text string from fast readoff, do the numpy conversion now
01572     if isinstance(frames[0], str):
01573         frames = generateFrames(frames)
01574
01575     framestemp = np.copy(frames)
01576
01577     for idx, frame in enumerate(framestemp):
01578         if idx < len(framestemp) / 2:
01579             interlacing = self.sensor.interlacing[0]
01580         else:
01581             interlacing = self.sensor.interlacing[1]
01582         try:
01583             if self.padToFull:
01584                 frame = np.reshape(
01585                     frame, (self.sensor.maxheight // (interlacing + 1), -1)
01586                 )
01587             else:
01588                 frame = np.reshape(
01589                     frame,
01590                     (
01591                         (self.sensor.lastrow - self.sensor.firstrow + 1)
01592                         // (interlacing + 1),
01593                         -1,
01594                     ),
01595                 )
01596             namenum = filename + "_%d" % firstnum
01597             nppath = os.path.join(path, prefix + namenum + ".npy")
01598             np.save(nppath, frame)
01599             firstnum += 1
01600         except SystemExit:
01601             raise
01602         except KeyboardInterrupt:
01603             raise
01604         except Exception:
01605             err = self.loggererr + "saveNumpys: unable to save arrays"
01606             logging.error(err)
01607             continue
01608     return err
01609
01610 def dumpNumpy(
01611     self,
01612     datastream,
01613     path=None,
01614     filename="Dump",
01615     prefix=None,
01616 ):
01617     """

```



```

01618         Datastream is converted directly to numpy array and saved to disk. No attempt to
01619         parse headers or separate into individual frames is made. The packet header is
01620         removed before saving
01621
01622     Args:
01623         datastream: string to be saved
01624         path: save path, defaults to './output'
01625         filename: defaults to 'Dump'
01626         prefix: prepended to 'filename', defaults to time/date
01627             (e.g. '160830-124704_')
01628
01629     Returns:
01630         Error string
01631     """
01632     logging.info(self.logininfo + "dumpNumpy")
01633     logging.debug(
01634         self.logdebug
01635         + "dumpNumpy: path = "
01636         + str(path)
01637         + "; filename = "
01638         + str(filename)
01639         + "; prefix = "
01640         + str(prefix)
01641     )
01642     err = ""
01643     if path is None:
01644         path = os.path.join(os.getcwd(), "output")
01645     if prefix is None:
01646         prefix = time.strftime("%y%m%d-%H%M%S_", time.localtime())
01647     if not os.path.exists(path):
01648         os.makedirs(path)
01649     npdata = str2npparray(datastream[36:])
01650     try:
01651         nppath = os.path.join(path, prefix + filename + ".npy")
01652         np.save(nppath, npdata)
01653     except SystemExit:
01654         raise
01655     except KeyboardInterrupt:
01656         raise
01657     except Exception:
01658         err = self.logerr + "dumpNumpy: unable to save data stream"
01659         logging.error(err)
01660     return err
01661
01662 def checkRegSet(self, regname, teststring):
01663     """
01664     Quick check to confirm that data read from register matches data write
01665
01666     Args:
01667         regname: register to test
01668         teststring: value to assign to register, as integer or hexadecimal string
01669             with or without '0x'
01670
01671     Returns:
01672         boolean, True if read and write values match
01673     """
01674     self.setRegister(regname, teststring)
01675     # tell board to send data; wait to clear before interrogating register contents
01676     if regname == "SRAM_CTL":
01677         time.sleep(2)
01678         if self.commname == "rs422":
01679             logging.info(
01680                 self.logininfo + "skipping 'SRAM_CTL' register check for RS422"
01681             )
01682             return True
01683         else:
01684             time.sleep(0.1)
01685         temp = self.getRegister(regname)
01686         resp = temp[1].upper()
01687         if resp != teststring.upper():
01688             logging.error(
01689                 self.logerr
01690                 + "checkRegSet failure: "
01691                 + regname
01692                 + " ; set: "
01693                 + teststring
01694                 + " ; read: "
01695                 + resp
01696             )
01697             return False
01698     return True

```

```

01699
01700 def initPowerCheck(self):
01701     """
01702     Reset software and board timers for monitoring power status
01703     """
01704     self.inittime = time.time()
01705     logging.info(self.logininfo + "resetting timer for power check function")
01706     self.resetTimer()
01707
01708 def powerCheck(self, delta=10):
01709     """
01710     Check to see if board power has persisted since powerCheck was last initialized.
01711     Compares time elapsed since initialization against board's timer. If the
01712     difference is greater than 'delta,' flag as False (power has likely failed)
01713
01714     Args:
01715         delta: difference in seconds permitted between software and board timers
01716
01717     Returns:
01718         boolean, 'True' means timer difference is less than 'delta' parameter;
01719         'False' indicates power failure
01720     """
01721     elapsed = time.time() - self.inittime
01722     logging.debug(self.logdebug + "powerCheck: elapsed time = " + str(elapsed))
01723     difference = abs(elapsed - self.getTimer())
01724     if difference > delta:
01725         logging.warning(
01726             self.logwarn + "powerCheck function has failed; may indicate current "
01727             "or recent power failure "
01728         )
01729     return difference < delta
01730
01731 def printBoardInfo(self):
01732     # TODO: add override option if logging level is above info
01733     logging.info(
01734         self.logininfo
01735         + "Python version: "
01736         + str(self.python)
01737         + "."
01738         + str(self.pyth1)
01739         + "."
01740         + str(self.pyth2)
01741     )
01742     logging.info(self.logininfo + "nsCamera software version: " + self.version)
01743     logging.info(self.logininfo + "FPGA firmware version: " + self.FPGAVersion)
01744     logging.info(self.logininfo + "FPGA implementation: " + self.FPGANum)
01745     if self.FPGAInvalid:
01746         logging.info(self.logininfo + "FPGA information unavailable")
01747     else:
01748         logging.info(self.logininfo + "Board type: " + self.FPGAboardtype)
01749         logging.info(self.logininfo + "Rad-Tolerant: " + str(self.FPGArad))
01750         logging.info(self.logininfo + "Sensor family: " + self.FPGAsensor)
01751         logging.info(self.logininfo + "Sensor label: " + self.sensor.loglabel)
01752         logging.info(
01753             self.logininfo + "Available interfaces: " + ", ".join(self.FPGAinterfaces)
01754         )
01755     if self.commname == "gige":
01756         ci = self.comms.CardInfoP.contents
01757         ip = ".".join(str(e) for e in [b for b in ci.IPAddr])
01758         logging.info(
01759             self.logininfo + "GigE connected to " + ip + ":" + str(self.port)
01760         )
01761     elif self.commname == "rs422":
01762         logging.info(self.logininfo + "RS422 connected to " + self.comms.port)
01763
01764 def dumpRegisters(self):
01765     """
01766     *DEPRECATED* use dumpStatus() instead
01767
01768     List contents of all registers in board.registers. *WARNING* some status flags
01769     will reset when read.
01770
01771     Returns:
01772         Sorted list: [register name (register address) : register contents as
01773         hexadecimal string without '0x']
01774     """
01775     dump = {}
01776     for key in self.board.registers.keys():
01777         err, rval = self.getRegister(key)
01778         dump[key] = rval
01779     reglistmax = int(max(self.board.registers.values()), 16)

```

```

01780         dumplist = [0] * (reglistmax + 1)
01781         for k, v in dump.items():
01782             regnum = self.board.registers[k]
01783             dumplist[int(regnum, 16)] = (
01784                 "(" + regnum + ") {0:<24} {1}".format(k, v.upper())
01785             )
01786         reglist = [a for a in dumplist if a]
01787         return reglist
01788
01789     def dumpSubregisters(self):
01790         """
01791         *DEPRECATED* use dumpStatus() instead
01792
01793         List contents of all subregisters in board.channel_lookups and
01794         board.monitor_lookups.
01795         *WARNING* some registers will reset when read; only the first subregister from
01796         such a register will return the correct value, the remainder will return zeros
01797
01798         Returns:
01799             dictionary {subregister name : subregister contents as binary string
01800                         without initial '0b'}
01801         """
01802         dump = {}
01803         for sub in self.board.subreglist:
01804             key = sub.name
01805             err, resp = self.getSubregister(key)
01806             if err:
01807                 logging.warning(
01808                     self.logwarn + "dumpSubregisters: unable to read subregister " + key
01809                 )
01810             val = hex(int(resp, 2))
01811             dump[key] = val
01812         return dump
01813
01814     def mmReadoff(self, waitOnSRAM, variation=None):
01815         """
01816         Convenience function for parsing frames for use by MicroManager plugin
01817         Args:
01818             waitOnSRAM: readoff wait flag
01819             variation: format of frames generated from readoff
01820                       default - return first frame only
01821                       "LastFrame" - return last frame only
01822                       "Average" - provide average of frames as single frame
01823                       "Landscape" - stitch frames together horizontally into single wide frame
01824
01825         Returns:
01826             ndarray - single image frame
01827         """
01828         frames, datalen, data_err = self.readoff(waitOnSRAM)
01829         if variation == "LastFrame":
01830             return frames[self.sensor.nframes - 1]
01831         elif variation == "Average":
01832             return np.sum(frames, axis=0) // self.sensor.nframes
01833         elif variation == "Landscape":
01834             shaped = [
01835                 np.reshape(frame, (self.sensor.maxheight, self.sensor.maxwidth))
01836                 for frame in frames
01837             ]
01838             return np.concatenate(shaped, axis=1)
01839         else:
01840             return frames[0]
01841
01842     def setFrame(self, minframe=None, maxframe=None):
01843         """
01844         Sets bounds on frames returned by board, inclusive (e.g., 0,3 returns four
01845         frames). If called without parameters, resets to full set of frames.
01846
01847         Args:
01848             minframe: first frame to read from board
01849             maxframe: last frame to read from board
01850
01851         Returns:
01852             Error string
01853         """
01854         logging.debug(
01855             self.logdebug
01856             + "setFrame: minframe = "
01857             + str(minframe)
01858             + "; maxframe = "
01859             + str(maxframe)
01860         )

```

```

01861         if minframe is None:
01862             minframe = self.sensor.minframe
01863         if maxframe is None:
01864             maxframe = self.sensor.maxframe
01865         if (
01866             not isinstance(minframe, int)
01867             or minframe < self.sensor.minframe
01868             or minframe > maxframe
01869             or not isinstance(maxframe, int)
01870             or maxframe > self.sensor.maxframe
01871         ):
01872             err = (
01873                 self.logerr + "setFrames: invalid frame limits submitted. Frame "
01874                 "selection remains unchanged. "
01875             )
01876             logging.error(err)
01877             return err
01878
01879         initframe = hex(minframe)[2:].zfill(8)
01880         finframe = hex(maxframe)[2:].zfill(8)
01881         err1, _ = self.setRegister("FPA_FRAME_INITIAL", initframe)
01882         err2, _ = self.setRegister("FPA_FRAME_FINAL", finframe)
01883         self.sensor.firstframe = minframe
01884         self.sensor.lastframe = maxframe
01885         self.sensor.nframes = maxframe - minframe + 1
01886         self.comms.payloadsize = (
01887             self.sensor.width
01888             * self.sensor.height
01889             * self.sensor.nframes
01890             * self.sensor.bytesperpixel
01891         )
01892         plural = ""
01893         if self.sensor.nframes > 1:
01894             plural = "s"
01895         logging.info(
01896             self.loginfo
01897             + "Readoff set to "
01898             + str(self.sensor.nframes)
01899             + " frame"
01900             + plural
01901             + " ("
01902             + str(minframe)
01903             + ", "
01904             + str(maxframe)
01905             + ")"
01906         )
01907         err = err1 + err2
01908         if err:
01909             logging.error(
01910                 self.logerr + "setFrames may not have functioned properly: " + err
01911             )
01912         return err
01913
01914 def setRows(self, minrow=0, maxrow=None, padToFull=False):
01915     """
01916     Sets bounds on rows returned by board, inclusive (e.g., 0,1023 returns all 1024
01917     rows). If called without parameters, resets to full image size.
01918
01919     Args:
01920         minrow: first row to return from board
01921         maxrow: last row to return from board
01922         padToFull: if True, generate full size frames, padding collected rows with
01923         zeroes if necessary
01924     """
01925     logging.debug(
01926         self.logdebug
01927         + "setRows: minrow = "
01928         + str(minrow)
01929         + "; maxrow = "
01930         + str(maxrow)
01931         + "; padToFull = "
01932         + str(padToFull)
01933     )
01934     if maxrow is None:
01935         maxrow = self.sensor.maxheight - 1
01936     if (
01937         not isinstance(minrow, int)
01938         or minrow < 0
01939         or minrow > maxrow
01940         or not isinstance(maxrow, int)
01941         or maxrow >= self.sensor.maxheight
    
```

```

01942         ):
01943             err = (
01944                 self.logger + "setRows: invalid row arguments submitted. Frame size"
01945                 " remains unchanged. "
01946             )
01947             logging.error(err)
01948             return err
01949
01950         initrow = hex(minrow)[2:].zfill(8)
01951         finrow = hex(maxrow)[2:].zfill(8)
01952         err1, _ = self.setRegister("FPA_ROW_INITIAL", initrow)
01953         err2, _ = self.setRegister("FPA_ROW_FINAL", finrow)
01954         self.sensor.firstrow = minrow
01955         self.sensor.lastrow = maxrow
01956         self.sensor.height = maxrow - minrow + 1
01957         self.comms.payloadsize = (
01958             self.sensor.width
01959             * self.sensor.height
01960             * self.sensor.nframes
01961             * self.sensor.bytesperpixel
01962         )
01963
01964         if self.commname == "rs422":
01965             self.comms.datatimeout = (
01966                 (1.0 * self.sensor.height / self.sensor.maxheight)
01967                 * 5e7
01968                 * self.sensor.nframes
01969                 / self.comms.baud
01970             )
01971         self.padToFull = padToFull
01972         logging.info(
01973             self.loginfo
01974             + "Readoff set to "
01975             + str(self.sensor.height)
01976             + " rows ("
01977             + str(minrow)
01978             + ", "
01979             + str(maxrow)
01980             + ") "
01981         )
01982         err = err1 + err2
01983         if err:
01984             logging.error(
01985                 self.logger + "setRows may not have functioned properly: " + err
01986             )
01987         return err
01988
01989     def abortReadoff(self, flag=True):
01990         """
01991         Simple abort command for readoff in waiting mode--does not interrupt download in
01992         progress. Requires external threading to function. *WARNING* if not
01993         intercepted by active readoff command, will terminate next readoff command
01994         immediately at inception.
01995         Args:
01996             flag: Sets passive abort flag read by readoff command
01997         Returns:
01998             boolean: updated setting of flag
01999         """
02000         logging.info(self.loginfo + "abortReadoff")
02001         self.abort = flag
02002         return flag
02003
02004     def batchAcquire(
02005         self,
02006         sets=1,
02007         trig="Hardware",
02008         path=None,
02009         filename="Frame",
02010         prefix=None,
02011         showProgress=0,
02012     ):
02013         """
02014         Acquire a series of images as fast as possible, then process and save to disk.
02015         *WARNING* This method stores images in RAM, so the number of sets that can be
02016         acquired in a single call is limited by available memory.
02017
02018         Args:
02019             sets: Number of acquisitions to perform
02020             trig: trigger type; 'hardware', 'software', or 'dual'
02021             path: save path, defaults to './output'
02022             filename: defaults to 'frames.bin'

```

```

02023         prefix: prepended to filename, defaults to time/date (e.g. '160830-124704_')
02024         DO NOT USE unless providing a varying value (a fixed prefix will cause
02025         overwriting)
02026         showProgress: if non-zero, show notice every 'showProgress' acquisitions and
02027         print total acquisition time
02028
02029     Returns:
02030         Time taken for acquisition (seconds)
02031     """
02032     logging.debug(
02033         self.logdebug
02034         + "batchAcquire: sets = "
02035         + str(sets)
02036         + "; trig = "
02037         + str(trig)
02038         + "; path = "
02039         + str(path)
02040         + "; filename = "
02041         + str(filename)
02042         + "; prefix = "
02043         + str(prefix)
02044         + "; showProgress = "
02045         + str(showProgress)
02046     )
02047     datalist = ["0"] * sets
02048     timelist = [datetime.now()] * sets
02049     logging.info(
02050         self.loginfo
02051         + "batchAcquire: temporarily disabling warning and information logging "
02052     )
02053     logging.getLogger().setLevel(self.verbmap.get(2))
02054     beforeread = time.time()
02055     for i in range(sets):
02056         if showProgress and not (i + 1) % showProgress:
02057             print(self.loginfo + "batchAcquire: Acquiring set " + str(i + 1))
02058             self.arm(trig)
02059             data, datalen, data_err = self.readoff(fast=True)
02060             datalist[i] = data
02061             timelist[i] = datetime.now()
02062     afterread = time.time()
02063     if showProgress:
02064         print(
02065             self.loginfo
02066             + "batchAcquire: "
02067             + str(afterread - beforeread)
02068             + " seconds for "
02069             + str(sets)
02070             + " sets"
02071         )
02072     setnum = 0
02073     if path is None:
02074         path = os.path.join(os.getcwd(), "output")
02075     for imset, imtime in zip(datalist, timelist):
02076         setnum = setnum + 1
02077         if showProgress and not setnum % showProgress:
02078             print(self.loginfo + "batchAcquire: Saving set " + str(setnum))
02079         parsed = generateFrames(self, imset)
02080         if prefix is None:
02081             setprefix = imtime.strftime("%Y%m%d-%H%M%S%f")[:-2] + "_"
02082         else:
02083             setprefix = prefix
02084         self.saveTiffs(parsed, path, filename, prefix=setprefix)
02085     logging.getLogger().setLevel(self.verblevel)
02086     logging.info(self.loginfo + "batchAcquire: re-enabling logging")
02087     return afterread - beforeread
02088
02089     # TODO: should this be just a flag for readoff instead of a distinct method?
02090     # TODO: make sure this handles single frames (list made already?), text frames
02091     # TODO: add documentation
02092     def saveHDF(
02093         self,
02094         frames,
02095         path=None,
02096         filename="Acquisition",
02097         prefix=None,
02098     ):
02099         """ """
02100         logging.info(self.loginfo + ": saveHDF")
02101         err = ""
02102         if path is None:
02103             path = os.path.join(os.getcwd(), "output")

```

```

02104         if prefix is None:
02105             prefix = datetime.now().strftime("%y%m%d-%H%M%S%f")[:-5] + "_"
02106         if not os.path.exists(path):
02107             os.makedirs(path)
02108
02109         h5file = os.path.join(path, prefix + filename + ".hdf5")
02110         with h5py.File(h5file, "w") as f:
02111             # shotgrp = f.create_group("DATA/SHOT")
02112             frame_index = 0
02113             for frame in frames:
02114                 grp = f.create_group("DATA/SHOT/FRAME_0" + str(frame_index))
02115                 data = grp.create_dataset(
02116                     "DATA", (self.sensor.height, self.sensor.width), data=frame
02117                 )
02118                 frame_index += 1
02119
02120
02121 """
02122 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
02123 LLNL-CODE-838080
02124
02125 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
02126 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
02127 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
02128 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
02129 be made under this license.
02130 """

```

7.17 C:/Users/hill35/git/camera_python/nsCamera/comms/GigE.py File Reference

Classes

- class [nsCamera.comms.GigE.GigE](#)
- class [nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO](#)

Namespaces

- namespace [nsCamera](#)
- namespace [nsCamera.comms](#)
- namespace [nsCamera.comms.GigE](#)

7.18 GigE.py

[Go to the documentation of this file.](#)

```

00001 # -*- coding: utf-8 -*-
00002 """
00003 Gigabit Ethernet interface for nsCamera.
00004
00005 Author: Jeremy Martin Hill (jerhill@llnl.gov)
00006
00007 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00008 LLNL-CODE-838080
00009
00010 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00011 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00012 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00013 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00014 be made under this license.
00015
00016 Version: 2.1.2 (February 2025)

```

```

00017 """
00018
00019 import ctypes as C
00020 import logging
00021 import os.path
00022 import sys
00023
00024 from nsCamera.utils.misc import generateFrames, str2bytes, bytes2str
00025
00026
00027 class GigE:
00028     """
00029     Code to manage Gigabit Ethernet connection to board. Each GigE object manages a
00030     single OT card; to use multiple cards, instantiate multiple cameraAssembler
00031     objects, each specifying the unique IPs of the corresponding OT card.
00032
00033     Note: Orange Tree card must be configured before use. See the README for details
00034
00035     Exposed methods:
00036     arm() - puts camera into wait state for external trigger
00037     readFrames() - waits for data ready register flag, then copies camera image data
00038     into numpy arrays
00039     readoff() - waits for data ready register flag, then copies camera image data
00040     into numpy arrays; returns payload, payload size, and error message
00041     sendCMD(pkt) - sends packet object via serial port
00042     readSerial(size, timeout) - read 'size' bytes from connection
00043     writeSerial(outstring) - submits string 'outstring' over connection
00044     closeDevice() - close connections and free resources
00045     getCardIP() - returns IP address of OT card
00046     getCardInfo() - prints report of details of OT card and connection
00047     """
00048
00049     def __init__(self, camassem):
00050         """
00051         Args:
00052             camassem: parent cameraAssembler object
00053         """
00054         self.ca = camassem
00055         self.logcrit = self.ca.logcritbase + "[GigE] "
00056         self.logerr = self.ca.logerrbase + "[GigE] "
00057         self.logwarn = self.ca.logwarnbase + "[GigE] "
00058         self.loginfo = self.ca.loginfobase + "[GigE] "
00059         self.logdebug = self.ca.logdebugbase + "[GigE] "
00060         logging.info(self.loginfo + "Initializing GigE comms object")
00061         self.mode = 1
00062         self.writeTimeout = 10000
00063         self.readTimeout = 10000
00064         self.payloadsize = (
00065             self.ca.sensor.width
00066             * self.ca.sensor.height
00067             * self.ca.sensor.nframes
00068             * self.ca.sensor.bytesperpixel
00069         )
00070         logging.debug(
00071             self.logdebug + "Payload size: " + str(self.payloadsize) + " bytes"
00072         )
00073         self.skipError = False
00074
00075         self.ZErrorDict = {
00076             0x8000: "Socket Error",
00077             0x8001: "Internal Error",
00078             0x8002: "Illegal Status Code",
00079             0x8003: "Null Parameter",
00080             0x8004: "Out of Memory",
00081             0x8005: "Invalid Connection Type",
00082             0x8006: "Illegal Connection",
00083             0x8007: "Socket Closed Unexpectedly",
00084             0x8008: "Timeout",
00085             0x8009: "Illegal Parameter",
00086         }
00087
00088         if self.ca.port:
00089             logging.debug(
00090                 self.logdebug + "Port supplied to GigE.py: " + str(self.ca.port)
00091             )
00092             if isinstance(self.ca.port, int) and 0 < self.ca.port < 65536:
00093                 self.dport = self.ca.port
00094             else:
00095                 logging.error(
00096                     self.logerr + "Invalid port number supplied, defaulting to "
00097                     "20482 "

```



```

00098         )
00099         self.dport = 20482
00100     else:
00101         self.dport = 20482 # default
00102
00103     self.ca.port = self.dport
00104     logging.debug(self.logdebug + "Port used by GigE.py: " + str(self.dport))
00105
00106     logging.debug(self.logdebug + "CPU architecture: " + str(self.ca.arch))
00107     if self.ca.arch == "64bit":
00108         arch = "64"
00109     else:
00110         arch = "32"
00111
00112     logging.debug(self.logdebug + "Operating system: " + str(self.ca.platform))
00113     if self.ca.platform == "Windows":
00114         lib_name = "ZestETM1.dll"
00115     elif self.ca.platform == "Linux" or self.ca.platform == "Darwin":
00116         lib_name = "libZestETM1.so"
00117     else:
00118         logging.warning(
00119             self.logwarn + "System does not self-identify as Linux, Windows, "
00120             "or Mac. Assuming posix-style libraries "
00121         )
00122         lib_name = "libZestETM1.so"
00123
00124     self.closecard = False
00125
00126     libpath = os.path.join(self.ca.packageroot, "comms", "ZestETM1", arch, lib_name)
00127     self._zest = C.DLL(libpath)
00128
00129     self.CardInfo = self.ZESTETM1_CARD_INFOZESTETM1_CARD_INFO()
00130     self.CardInfoP = C.pointer(self.CardInfo)
00131
00132     # functions
00133     self.ZCountCards = self._zest.ZestETM1CountCards
00134     self.ZCountCards.argtypes = [
00135         C.POINTER(C.c_ulong),
00136         C.POINTER(C.POINTER(self.ZESTETM1_CARD_INFOZESTETM1_CARD_INFO)),
00137         C.c_int,
00138     ]
00139
00140     self.ZOpenConnection = self._zest.ZestETM1OpenConnection
00141     self.ZOpenConnection.argtypes = [
00142         C.POINTER(self.ZESTETM1_CARD_INFOZESTETM1_CARD_INFO),
00143         C.c_int,
00144         C.c_ushort,
00145         C.c_ushort,
00146         C.POINTER(C.c_void_p),
00147     ]
00148
00149     self.ZWriteData = self._zest.ZestETM1WriteData
00150     self.ZWriteData.argtypes = [
00151         C.c_void_p,
00152         C.c_void_p,
00153         C.c_ulong,
00154         C.POINTER(C.c_ulong),
00155         C.c_ulong,
00156     ]
00157
00158     self.ZReadData = self._zest.ZestETM1ReadData
00159     self.ZReadData.argtypes = [
00160         C.c_void_p,
00161         C.c_void_p,
00162         C.c_ulong,
00163         C.POINTER(C.c_ulong),
00164         C.c_ulong,
00165     ]
00166
00167     self.Connection = C.c_void_p()
00168     self.openDevice(self.ca.timeout)
00169
00170     def sendCMD(self, pkt):
00171         """
00172         Submit packet and verify the response packet.
00173         Packet communications with FPGA omit CRC suffix, so adds fake CRC bytes to
00174         response
00175
00176         Args:
00177             pkt: Packet object
00178

```

```

00179         Returns:
00180             tuple (error, response string)
00181         """
00182
00183         pktStr = pkt.pktStr()[0:16]
00184         logging.debug(self.logdebug + "sendCMD packet: " + str(pktStr))
00185         err = ""
00186         self.ca.writeSerial(pktStr)
00187         if (
00188             hasattr(self.ca, "board")
00189             and pktStr[4] == "0"
00190             and pktStr[5:8] == self.ca.board.registers["SRAM_CTL"]
00191         ):
00192             bufsize = self.payloadsize + 16
00193             resptext = self.readSerial(bufsize)
00194
00195             if len(resptext) < 32:
00196                 logging.debug(self.logdebug + "sendCMD resptext = " + str(resptext))
00197             else:
00198                 logging.debug(
00199                     self.logdebug
00200                     + "sendCMD resptext (truncated) = "
00201                     + str(resptext)[0:32]
00202                 )
00203
00204             if len(resptext) < bufsize + 16:
00205                 err += (
00206                     self.logerr + "sendCMD- packet too small, payload may be incomplete"
00207                 )
00208                 logging.error(err)
00209             else:
00210                 # workaround for initial setup before board object has been initialized
00211                 resp = self.readSerial(8)
00212                 logging.debug(self.logdebug + "sendCMD response: " + str(resp))
00213                 if len(resp) < 8:
00214                     err += self.logerr + "sendCMD- response too small, returning zeros"
00215                     resptext = "00000000000000000000"
00216                     logging.error(err)
00217                 else:
00218                     resptext = resp + "0000"
00219             return err, resptext
00220
00221 def arm(self, mode):
00222     """
00223     Puts camera into wait state for trigger. Mode determines source; defaults to
00224     'Hardware'
00225
00226     Args:
00227         mode: 'Software'|'S' activates software, disables hardware triggering
00228              'Hardware'|'H' activates hardware, disables software triggering
00229              Hardware is the default
00230
00231     Returns:
00232         tuple (error, response string)
00233     """
00234     if not mode:
00235         mode = "Hardware"
00236         logging.info(self.loginfo + "arm")
00237         logging.debug(self.logdebug + "arming mode: " + str(mode))
00238         self.ca.clearStatus()
00239         self.ca.latchPots()
00240         err, resp = self.ca.startCapture(mode)
00241         if err:
00242             logging.error(self.logerr + "unable to arm camera")
00243         else:
00244             self.ca.armed = True
00245             self.skipError = True
00246         return err, resp
00247
00248 def readFrames(self, waitOnSRAM, timeout=0, fast=False, columns=1):
00249     """
00250     Copies image data from board into numpy arrays.
00251
00252     Args:
00253         waitOnSRAM: if True, wait until SRAM_READY flag is asserted to begin copying
00254                     data
00255         timeout: passed to waitForSRAM; after this many seconds begin copying data
00256                 irrespective of SRAM_READY status; 'zero' means wait indefinitely
00257                 WARNING: If acquisition fails, the SRAM will not contain a current image,
00258                         but the code will copy the data anyway
00259         fast: if False, parse and convert frames to numpy arrays; if True, return

```

```

00260         unprocessed text stream
00261         columns: 1 for single image per frame, 2 for separate hemisphere images
00262
00263     Returns:
00264         list of numpy arrays OR raw text stream
00265
00266     """
00267     frames, _, _ = self.readoff(waitOnSRAM, timeout, fast, columns)
00268     return frames
00269
00270 def readoff(self, waitOnSRAM, timeout=0, fast=False, columns=1):
00271     """
00272     Copies image data from board into numpy arrays; returns data, length of data,
00273     and error messages. Use 'readFrames()' unless you require this additional
00274     information
00275
00276     Args:
00277         waitOnSRAM: if True, wait until SRAM_READY flag is asserted to begin copying
00278         data
00279         timeout: passed to waitForSRAM; after this many seconds begin copying data
00280         irrespective of SRAM_READY status; 'zero' means wait indefinitely
00281         WARNING: If acquisition fails, the SRAM will not contain a current image,
00282         but the code will copy the data anyway
00283         fast: if False, parse and convert frames to numpy arrays; if True, return
00284         unprocessed text stream
00285         columns: 1 for single image per frame, 2 for separate hemisphere images
00286
00287     Returns:
00288         tuple (list of numpy arrays OR raw text stream, length of downloaded payload
00289         in bytes, payload error flag) since CRC check is handled by TCP/IP,
00290         payload error flag is always False for GigE
00291     """
00292     logging.info(self.loginfo + "readoff")
00293     logging.debug(
00294         self.logdebug
00295         + "readoff: waitonSRAM = "
00296         + str(waitOnSRAM)
00297         + "; timeout = "
00298         + str(timeout)
00299         + "; fast = "
00300         + str(fast)
00301     )
00302     # Wait for data to be ready on board
00303     # Skip wait only if explicitly tagged 'False' ('None' defaults to True)
00304     if waitOnSRAM is not False:
00305         self.ca.waitForSRAM(timeout)
00306     self.skipError = False
00307     err, rval = self.ca.readSRAM()
00308     if err:
00309         logging.error(self.logerr + "Error detected in readSRAM")
00310     elif self.ca.boardname == "l1nl_v4":
00311         # self.ca.setSubregister('SWACK','1')
00312         pass
00313     # extract the data. Remove header; the FPGA returns a packet without the CRC
00314     # suffix
00315     logging.debug(self.logdebug + "readoff: first 64 chars: " + str(rval[0:64]))
00316     data = rval[32:]
00317     if fast:
00318         return data, len(data) // 2, bool(err)
00319     else:
00320         parsed = generateFrames(self.ca, data, columns)
00321         return parsed, len(data) // 2, bool(err)
00322
00323 def writeSerial(self, outstring, timeout):
00324     """
00325     Transmit string to board
00326
00327     Args:
00328         outstring: string to write
00329         timeout: serial timeout in sec (defaults to self.writeTimeout)
00330
00331     Returns:
00332         integer number of bytes written
00333     """
00334     logging.debug(
00335         self.logdebug
00336         + "writeSerial: outstring = "
00337         + str(outstring)
00338         + "; timeout = "
00339         + str(timeout)
00340     )

```

```

00341         if not timeout:
00342             timeout = self.writeTimeout
00343         outstring = str2bytes(outstring)
00344         outbuff = C.create_string_buffer(outstring)
00345         outbuffp = C.pointer(outbuff)
00346         outbufflen = len(outstring)
00347         writelen = C.c_ulong(0)
00348         err = self.ZWriteData(
00349             self.Connection, outbuffp, outbufflen, C.byref(writelen), timeout
00350         )
00351         if err:
00352             if err == 0x4000:
00353                 logging.warning(
00354                     self.logerr + "OT Card emitted an undefined warning message"
00355                 )
00356             else:
00357                 logging.error(
00358                     self.logerr + "writeSerial error: " + self.ZErrorDict[err]
00359                 )
00360             logging.debug(self.logdebug + "writeSerial: writelen = " + str(writelen))
00361             return writelen
00362
00363     def readSerial(self, size, timeout=None):
00364         """
00365         Read bytes from the serial port. Does not verify packets.
00366
00367         Args:
00368             size: number of bytes to read
00369             timeout: serial timeout in sec (defaults to self.readTimeout)
00370
00371         Returns:
00372             tuple (error string, string read from serial port)
00373         """
00374         logging.debug(
00375             self.logdebug
00376             + "readSerial: size = "
00377             + str(size)
00378             + "; timeout = "
00379             + str(timeout)
00380         )
00381         if not timeout:
00382             timeout = self.readTimeout
00383         inbuff = C.create_string_buffer(size + 1)
00384         inbuffp = C.pointer(inbuff)
00385         readlen = C.c_ulong(0)
00386         err = self.ZReadData(self.Connection, inbuffp, size, C.byref(readlen), timeout)
00387         if err:
00388             if self.skipError:
00389                 logging.debug(
00390                     self.logdebug + "readSerial: skipped error: " + self.ZErrorDict[err]
00391                 )
00392                 self.skipError = False
00393             elif err == 0x4000:
00394                 logging.warning(
00395                     self.logerr + "OT Card emitted an undefined warning message"
00396                 )
00397             else:
00398                 logging.error(self.logerr + "readSerial error: " + self.ZErrorDict[err])
00399             return bytes2str(inbuff.raw)[:2]
00400
00401     # TODO: check for valid timeout, probably in init in CameraAssembler
00402     def openDevice(self, timeout=30):
00403         """
00404         Find Orange Tree card and open a connection; if IP is supplied as parameter for
00405         the CameraAssembler, bypass network search and connect directly to indicated
00406         IP address
00407
00408         Args:
00409             timeout: timeout in seconds for attempting to connect to a card
00410         """
00411         err = self._zest.ZestETM1Init()
00412         if err:
00413             logging.critical(self.logcrit + "ZestETM1Init failure")
00414             sys.exit(1)
00415         logging.info(self.loginf + "searching for Orange Tree cards")
00416         NumCards = C.c_ulong(0)
00417
00418         if self.ca.iplist:
00419             ubyte4 = C.c_ubyte * 4
00420             self.CardInfo.IPAddr = ubyte4(*self.ca.iplist)
00421             self.CardInfo.ControlPort = C.c_ushort(self.dport)

```

```

00422         self.CardInfo.Timeout = C.c_ulong(self.writeTimeout)
00423         self.closecard = False
00424     else:
00425         wait = 0
00426         while True:
00427             logging.debug(
00428                 self.logdebug + "openDevice: connection wait = " + str(wait)
00429             )
00430             err = self.ZCountCards(C.byref(NumCards), C.byref(self.CardInfoP), 1000)
00431             if err:
00432                 logging.critical(self.logcrit + "CountCards failure")
00433                 sys.exit(1)
00434             if NumCards.value > 0:
00435                 break
00436             if wait == timeout:
00437                 logging.critical(
00438                     "{}No Orange Tree cards found in {} seconds".format(
00439                         self.logcrit, timeout
00440                     )
00441                 )
00442                 sys.exit(1)
00443             elif not wait % 5:
00444                 logging.info(
00445                     "{}Still trying to connect after {} seconds...".format(
00446                         self.loginfo, wait
00447                     )
00448                 )
00449                 wait += 1
00450             logging.info(
00451                 self.loginfo + str(NumCards.value) + " Orange Tree card(s) found"
00452             ) # TODO: add check for GigE bit in board description
00453         err = self.ZOpenConnection(
00454             self.CardInfoP, 0, self.dport, 0, C.byref(self.Connection)
00455         )
00456         if err:
00457             if err == 0x4000:
00458                 logging.warning(
00459                     self.logerr + "OT Card emitted an undefined warning message"
00460                 )
00461             else:
00462                 logging.critical(
00463                     self.logcrit + "OpenConnection failure: " + self.ZErrorDict[err]
00464                 )
00465             sys.exit(1)
00466
00467     def closeDevice(self):
00468         """
00469         Close connection to Orange Tree card and free resources
00470         """
00471         logging.debug(self.logdebug + "Closing connection to Orange Tree card")
00472         self._zest.ZestETM1CloseConnection(self.Connection)
00473         if self.closecard:
00474             try:
00475                 self._zest.ZestETM1FreeCards(self.CardInfoP)
00476             except SystemExit:
00477                 raise
00478             except KeyboardInterrupt:
00479                 raise
00480             except Exception:
00481                 logging.error(self.logerr + "Error reported in OT card closure")
00482         self._zest.ZestETM1Close()
00483
00484     def getCardIP(self):
00485         """
00486         Query IP address of OT card
00487
00488         Returns: address of OT card as list of bytes
00489         """
00490         return self.CardInfo.IPAddr
00491
00492     # TODO: use logging.info, with override option?
00493     def getCardInfo(self):
00494         """
00495         Prints status message with information returned by OT card
00496         """
00497         ci = self.CardInfoP.contents
00498         print("GigE Card Status:")
00499         print("-----")
00500         print("IP: " + " ".join(str(e) for e in [b for b in ci.IPAddr]))
00501         print("ControlPort: " + str(ci.ControlPort))
00502         print("Timeout: " + str(ci.Timeout))

```

```

00503     print("HTTPPort: " + str(ci.HTTPPort))
00504     print("MACAddr: " + ".".join(format(e, "02X") for e in [b for b in ci.MACAddr]))
00505     print("SubNet: " + ".".join(str(e) for e in [b for b in ci.SubNet]))
00506     print("Gateway: " + ".".join(str(e) for e in [b for b in ci.Gateway]))
00507     print("SerialNumber: " + str(ci.SerialNumber))
00508     print("FirmwareVersion: " + str(ci.FirmwareVersion))
00509     print("HardwareVersion: " + str(ci.HardwareVersion))
00510     print("-----")
00511
00512     class ZESTETM1_CARD_INFO(C.Structure):
00513         ubyte4 = C.c_ubyte * 4
00514         ubyte6 = C.c_ubyte * 6
00515         _fields_ = [
00516             ("IPAddr", ubyte4),
00517             ("ControlPort", C.c_ushort),
00518             ("Timeout", C.c_ulong),
00519             ("HTTPPort", C.c_ushort),
00520             ("MACAddr", ubyte6),
00521             ("SubNet", ubyte4),
00522             ("Gateway", ubyte4),
00523             ("SerialNumber", C.c_ulong),
00524             ("FirmwareVersion", C.c_ulong),
00525             ("HardwareVersion", C.c_ulong),
00526         ]
00527
00528
00529     """
00530 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00531 LLNL-CODE-838080
00532
00533 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00534 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00535 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00536 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00537 be made under this license.
00538 """

```

7.19 C:/Users/hill35/git/camera_python/nsCamera/comms/RS422.py File Reference

Classes

- class [nsCamera.comms.RS422.RS422](#)

Namespaces

- namespace [nsCamera](#)
- namespace [nsCamera.comms](#)
- namespace [nsCamera.comms.RS422](#)

7.20 RS422.py

[Go to the documentation of this file.](#)

```

00001 # -*- coding: utf-8 -*-
00002 """
00003 RS422 driver for nsCamera
00004
00005 Author: Brad Funsten (funsten1@llnl.gov)
00006 Author: Jeremy Martin Hill (jerhill@llnl.gov)
00007
00008 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00009 LLNL-CODE-838080

```

```

00010
00011 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00012 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00013 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00014 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00015 be made under this license.
00016
00017 Version: 2.1.2 (February 2025)
00018 """
00019
00020 import logging
00021 import sys
00022 import time # to time the script
00023
00024 import serial
00025 import serial.tools.list_ports # for RS422 serial link setup
00026
00027 from nsCamera.utils.misc import generateFrames, str2bytes, bytes2str, checkCRC
00028
00029
00030 class RS422:
00031     """
00032     Code to manage RS422 connection. Will automatically query available COM interfaces
00033     until a board is found. Use the 'port=x' parameter in cameraAssembler call to
00034     specify a particular COM interface.
00035
00036     Exposed methods:
00037     arm() - Puts camera into wait state for external trigger
00038     readFrames() - waits for data ready register flag, then copies camera image data
00039         into numpy arrays
00040     readoff() - waits for data ready register flag, then copies camera image data
00041         into numpy arrays; returns payload, payload size, and error message
00042     sendCMD(pkt) - sends packet object via serial port
00043     readSerial(size, timeout) - read 'size' bytes from serial port
00044     writeSerial(cmd) - submits string 'cmd' (assumes string is preformed packet)
00045     closeDevice() - close serial connections
00046     """
00047
00048     def __init__(self, camassem, baud=921600, par="O", stop=1):
00049         """
00050         Args:
00051             camassem: parent cameraAssembler object
00052             baud: bits per second
00053             par: parity type
00054             stop: number of stop bits
00055         """
00056         self.ca = camassem
00057         self.logcrit = self.ca.logcritbase + "[RS422] "
00058         self.logerr = self.ca.logerrbase + "[RS422] "
00059         self.logwarn = self.ca.logwarnbase + "[RS422] "
00060         self.loginfo = self.ca.loginfobase + "[RS422] "
00061         self.logdebug = self.ca.logdebugbase + "[RS422] "
00062         logging.info(self.loginfo + "initializing RS422 comms object")
00063         logging.debug(
00064             self.logdebug
00065             + "Init: baud = "
00066             + str(baud)
00067             + "; par = "
00068             + str(par)
00069             + "; stop = "
00070             + str(stop)
00071         )
00072         self.mode = 0
00073         self.baud = baud # Baud rate (bits/second)
00074         self.par = par # Parity bit
00075         self.stop = stop # Number of stop bits
00076         self.read_timeout = 1 # default timeout for ordinary packets
00077         self.write_timeout = 1
00078         # TODO: make datatimeout a cameraAssembler parameter
00079         self.datatimeout = 60 # timeout for data read
00080         logging.debug(
00081             self.logdebug + "Data timeout = " + str(self.datatimeout) + " seconds"
00082         )
00083         self.PY3 = sys.version_info > (3,)
00084         self.skipError = False
00085         port = ""
00086         ports = list(serial.tools.list_ports.comports())
00087         logging.debug(self.logdebug + "Comports: " + str(ports))
00088         for p, desc, add in ports:
00089             if self.ca.port is None or p == "COM" + str(self.ca.port):
00090                 logging.info(self.loginfo + "found comm port " + p)

```

```

00091         try:
00092             with serial.Serial(
00093                 p,
00094                 self.baud,
00095                 parity=self.par,
00096                 timeout=0.01,
00097                 write_timeout=0.01,
00098             ) as ser:
00099                 ser.write(str2bytes("aaaa10000000000001a84"))
00100                 time.sleep(1)
00101                 s = ser.read(10)
00102                 resp = bytes2str(s)
00103                 logging.debug(self.logdebug + "Init response: " + str(resp))
00104                 if (
00105                     resp[0:5].lower() == "aaaa9"
00106                 ): # TODO: add check for RS422 bit in board description
00107                     boardid = resp[8:10]
00108                     if boardid == "00":
00109                         logging.critical(
00110                             self.logcrit + "SNLrevC board detected - not "
00111                             "compatible with nsCamera >= 2.0"
00112                         )
00113                         sys.exit(1)
00114                     elif boardid == "81":
00115                         logging.info(self.loginfo + "LLNLv1 board detected")
00116                     elif boardid == "84":
00117                         logging.info(self.loginfo + "LLNLv4 board detected")
00118                     else:
00119                         logging.info(
00120                             self.loginfo + "unidentified board detected"
00121                         )
00122                         logging.info(self.loginfo + "connected to " + p)
00123                         port = p
00124                         ser.reset_input_buffer()
00125                         ser.reset_output_buffer()
00126                         break
00127             except Exception as e:
00128                 logging.error(self.logerr + "port identification: " + str(e))
00129         if port == "":
00130             if self.ca.port:
00131                 logging.critical(
00132                     self.logcrit + "No usable board found at port " + str(self.ca.port)
00133                 )
00134                 sys.exit(1)
00135             else:
00136                 logging.critical(self.logcrit + "No usable board found")
00137                 sys.exit(1)
00138         self.port = port # COM port to use for RS422 link
00139         self.ca.port = port[3:] # re-extract port number from com name
00140
00141         self._ser = serial.Serial( # Class RS422
00142             port=self.port,
00143             baudrate=self.baud,
00144             parity=self.par,
00145             stopbits=self.stop,
00146             timeout=self.read_timeout, # timeout for serial read
00147             bytesize=serial.EIGHTBITS,
00148         )
00149         self.payloadsize = (
00150             self.ca.sensor.width
00151             * self.ca.sensor.height
00152             * self.ca.sensor.nframes
00153             * self.ca.sensor.bytesperpixel
00154         )
00155         logging.debug(
00156             self.logdebug + "Payload size: " + str(self.payloadsize) + " bytes"
00157         )
00158         self._ser.flushInput()
00159         if not self._ser.is_open:
00160             logging.critical(self.logcrit + "Unable to open serial connection")
00161             sys.exit(1)
00162
00163     def serialClose(self):
00164         """
00165         Close serial interface
00166         """
00167         logging.debug(self.logdebug + "serialclose")
00168         self._ser.close() # close serial interface COM port
00169
00170     def sendCMD(self, pkt):
00171         """

```



```

00172         Submit packet and verify response packet. Recognizes readoff packet and adjusts
00173         read size and timeout appropriately
00174
00175     Args:
00176         pkt: Packet object
00177
00178     Returns:
00179         tuple (error, response string)
00180
00181     """
00182     pktStr = pkt.pktStr()
00183     logging.debug(self.logdebug + "sendCMD packet: " + str(pktStr))
00184     self._ser.flushInput()
00185     time.sleep(0.01) # wait 10 ms in between flushing input and output buffers
00186     self._ser.flushOutput()
00187     self.ca.writeSerial(pktStr)
00188     err0 = ""
00189     err = ""
00190     resp = ""
00191     tries = 3 # TODO: make a function parameter?
00192
00193     if (
00194         hasattr(self.ca, "board")
00195         and pktStr[4] == "0"
00196         and pktStr[5:8] == self.ca.board.registers["SRAM_CTL"]
00197     ):
00198         # download data payload
00199         logging.info(
00200             self.loginfo + "Payload size (bytes) = " + str(self.payloadsize)
00201         )
00202         crcresp0 = ""
00203         crcresp1 = ""
00204         smallresp = ""
00205         emptyResponse = False
00206         wrongSize = False
00207         # TODO: refactor payload error management to another method
00208         for i in range(tries):
00209             err, resp = self.readSerial(
00210                 self.payloadsize + 20, timeout=self.datatimeout
00211             )
00212             if err:
00213                 logging.error(
00214                     self.logerr + "sendCMD: read payload failed " + pktStr + err
00215                 )
00216                 self.ca.payloaderror = True
00217             else:
00218                 if not len(resp):
00219                     err0 = self.logerr + "sendCMD: empty response from board"
00220                     logging.error(err0)
00221                     emptyResponse = True
00222                     self.ca.payloaderror = True
00223                 elif len(resp) != 2 * (self.payloadsize + 20):
00224                     err0 = (
00225                         self.logerr
00226                         + "sendCMD: incorrect response; expected "
00227                         + str(self.payloadsize + 20)
00228                         + " bytes, received "
00229                         + str(len(resp) // 2)
00230                     )
00231                     logging.error(err0)
00232                     wrongSize = True
00233                     smallresp = resp
00234                     self.ca.payloaderror = True
00235                 elif not checkCRC(resp[4:20]):
00236                     err0 = (
00237                         self.logerr
00238                         + "sendCMD: "
00239                         + pktStr
00240                         + " - payload preface CRC fail"
00241                     )
00242                     logging.error(err0)
00243                     self.ca.payloaderror = True
00244                     crcresp1 = resp
00245                 elif not checkCRC(resp[24:]):
00246                     err0 = (
00247                         self.logerr + "sendCMD: " + pktStr + " - payload CRC fail"
00248                     )
00249                     logging.error(err0)
00250                     self.ca.payloaderror = True
00251                     crcresp0 = resp
00252             err += err0
00253             time.sleep(5)

```

```

00253         if self.ca.payloaderror:
00254             # keep best results over multiple tries; e.g., if first try is
00255             # bad CRC and second try is an incomplete payload, use the
00256             # first payload
00257             if i == tries - 1:
00258                 if crcresp0:
00259                     logging.error(
00260                         self.logerr + "sendCMD: Unable to acquire "
00261                         "CRC-confirmed payload after "
00262                         + str(tries)
00263                         + " attempts. Continuing with unconfirmed payload"
00264                     )
00265                     resp = crcresp0
00266                 elif crcresp1:
00267                     logging.error(
00268                         self.logerr + "sendCMD: Unable to acquire "
00269                         "CRC-confirmed readoff after "
00270                         + str(tries)
00271                         + " attempts. Continuing with unconfirmed payload"
00272                     )
00273                     resp = crcresp1
00274                 elif wrongSize:
00275                     logging.error(
00276                         self.logerr + "sendCMD: Unable to acquire complete "
00277                         "payload after "
00278                         + str(tries)
00279                         + " attempts. Dumping datastream to file."
00280                     )
00281                     resp = smallresp
00282                     self.ca.dumpNumpy(resp)
00283                 elif emptyResponse:
00284                     logging.error(
00285                         self.logerr + "sendCMD: Unable to acquire any "
00286                         "payload after " + str(tries) + " attempts."
00287                     )
00288             else:
00289                 logging.info(
00290                     self.loginfo + "Retrying download, attempt #" + str(i + 1)
00291                 )
00292                 err = ""
00293                 err0 = ""
00294                 self.ca.payloaderror = False
00295                 self.ca.writeSerial(pktStr)
00296             else:
00297                 logging.info(self.loginfo + "Download successful")
00298                 if self.ca.boardname == "llnl_v4":
00299                     # self.ca.setSubregister('SWACK','1')
00300                     pass
00301                 break
00302
00303     else:
00304         # non-payload messages and workaround for initial setup before board object
00305         # has been initialized
00306         time.sleep(0.03)
00307         self._ser.timeout = 0.02
00308         err, resp = self.readSerial(10)
00309         logging.debug(self.logdebug + "sendCMD response: " + str(resp))
00310         if err:
00311             logging.error(
00312                 self.logerr + "sendCMD: readSerial failed (regular packet) " + err
00313             )
00314         elif not checkCRC(resp[4:20]):
00315             err = self.logerr + "sendCMD- regular packet CRC fail: " + resp
00316             logging.error(err)
00317         return err, resp
00318
00319     def arm(self, mode):
00320         """
00321         Puts camera into wait state for trigger. Mode determines source; defaults to
00322         'Hardware'
00323
00324         Args:
00325             mode: 'Software'/'S' activates software, disables hardware triggering
00326                  'Hardware'/'H' activates hardware, disables software triggering
00327                  Hardware is the default
00328
00329         Returns:
00330             tuple (error, response string)
00331         """
00332         if not mode:
00333             mode = "Hardware"

```

```

00334         logging.info(self.loginfo + "arm")
00335         logging.debug(self.logdebug + "arming mode: " + str(mode))
00336         self.ca.clearStatus()
00337         self.ca.latchPots()
00338         err, resp = self.ca.startCapture(mode)
00339         if err:
00340             logging.error(self.logerr + "unable to arm camera")
00341         else:
00342             self.ca.armed = True
00343             self.skipError = True
00344         return err, resp
00345
00346     def readFrames(self, waitOnSRAM, timeout=0, fast=False, columns=1):
00347         """
00348         Copies image data from board into numpy arrays.
00349
00350         Args:
00351             waitOnSRAM: if True, wait until SRAM_READY flag is asserted to begin copying
00352                         data
00353             timeout: passed to waitForSRAM; after this many seconds begin copying data
00354                     irrespective of SRAM_READY status; 'zero' means wait indefinitely
00355                     WARNING: If acquisition fails, the SRAM will not contain a current image,
00356                             but the code will copy the data anyway
00357             fast: if False, parse and convert frames to numpy arrays; if True, return
00358                  unprocessed text stream
00359             columns: 1 for single image per frame, 2 for separate hemisphere images
00360
00361         Returns:
00362             list of numpy arrays OR raw text stream
00363
00364         """
00365         frames, _, _ = self.readoff(waitOnSRAM, timeout, fast, columns)
00366         return frames
00367
00368     def readoff(self, waitOnSRAM, timeout, fast, columns=1):
00369         """
00370         Copies image data from board into numpy arrays; returns data, length of data,
00371         and error messages. Use 'readFrames()' unless you require this additional
00372         information
00373
00374         Args:
00375             waitOnSRAM: if True, wait until SRAM_READY flag is asserted to begin copying
00376                         data
00377             timeout: passed to waitForSRAM; after this many seconds begin copying data
00378                     irrespective of SRAM_READY status; 'zero' means wait indefinitely
00379                     WARNING: If acquisition fails, the SRAM will not contain a current image,
00380                             but the code will copy the data anyway
00381             fast: if False, parse and convert frames to numpy arrays; if True, return
00382                  unprocessed text stream
00383             columns: 1 for single image per frame, 2 for separate hemisphere images
00384
00385         Returns:
00386             tuple (list of numpy arrays OR raw text stream, length of downloaded payload
00387                    in bytes, payload error flag)
00388             NOTE: This reduces readoff by <1 second, so will have no noticeable impact
00389                   when using RS422
00390
00391         """
00392         logging.info(self.loginfo + "readoff")
00393         logging.debug(
00394             self.logdebug
00395             + "readoff: waitonSRAM = "
00396             + str(waitOnSRAM)
00397             + "; timeout = "
00398             + str(timeout)
00399             + "; fast = "
00400             + str(fast)
00401         )
00402         errortemp = False
00403
00404         # Wait for data to be ready on board, turns off error messaging
00405         # Skip wait only if explicitly tagged 'False' ('None' defaults to True)
00406         if waitOnSRAM is not False:
00407             logging.getLogger().setLevel(logging.CRITICAL)
00408             self.ca.waitForSRAM(timeout)
00409             logging.getLogger().setLevel(self.ca.verblevel)
00410
00411         # Retrieve data
00412         err, rval = self.ca.readSRAM()
00413         if err:
00414             logging.error(self.logerr + "Error detected in readSRAM")
00415             time.sleep(0.3)

```

```

00415         logging.debug(self.logdebug + "readoff: first 64 chars: " + str(rval[0:64]))
00416         # extract only the read burst data. Remove header & CRC footer
00417         read_burst_data = rval[36:-4]
00418
00419         # Payload size as string implied by provided parameters
00420         expectedlength = (
00421             4
00422             * (self.ca.sensor.lastframe - self.ca.sensor.firstframe + 1)
00423             * (self.ca.sensor.lastrow - self.ca.sensor.firstrow + 1)
00424             * self.ca.sensor.width
00425         )
00426         padding = expectedlength - len(read_burst_data)
00427         if padding:
00428             logging.warning(
00429                 "{logwarn}readoff: Payload is shorter than expected."
00430                 " Padding with '0's".format(logwarn=self.logwarn)
00431             )
00432             read_burst_data = read_burst_data.ljust(expectedlength, "0")
00433
00434         if fast:
00435             return read_burst_data, len(read_burst_data) // 2, errortemp
00436         else:
00437             parsed = generateFrames(self.ca, read_burst_data, columns)
00438             return parsed, len(read_burst_data) // 2, errortemp
00439
00440     def writeSerial(self, outstring, timeout):
00441         """
00442         Transmit string to board
00443
00444         Args:
00445             outstring: string to write
00446             timeout: serial timeout in sec
00447         Returns:
00448             integer length of string written to serial port
00449         """
00450         logging.debug(
00451             self.logdebug
00452             + "writeSerial: outstring = "
00453             + str(outstring)
00454             + "; timeout = "
00455             + str(timeout)
00456         )
00457         if timeout:
00458             self.__ser.timeout = timeout
00459         else:
00460             self.__ser.timeout = self.write_timeout
00461         lengthwritten = self.__ser.write(str2bytes(outstring))
00462         self.__ser.timeout = self.read_timeout # reset if changed above
00463         return lengthwritten
00464
00465     def readSerial(self, size, timeout=None):
00466         """
00467         Read bytes from the serial port. Does not verify packets.
00468
00469         Args:
00470             size: number of bytes to read
00471             timeout: serial timeout in sec
00472
00473         Returns:
00474             tuple (error string, string read from serial port)
00475         """
00476         logging.debug(
00477             self.logdebug
00478             + "readSerial: size = "
00479             + str(size)
00480             + "; timeout = "
00481             + str(timeout)
00482         )
00483         err = ""
00484         if timeout:
00485             self.__ser.timeout = timeout
00486         else:
00487             self.__ser.timeout = self.read_timeout
00488         resp = self.__ser.read(size)
00489         if len(resp) < 10: # bytes
00490             err += (
00491                 self.logerr + "readSerial : packet too small: '" + bytes2str(resp) + "'"
00492             )
00493             logging.error(err)
00494         return err, bytes2str(resp)
00495

```

```

00496     def closeDevice(self):
00497         """
00498         Close primary serial interface
00499         """
00500         logging.debug(self.logdebug + "Closing RS422 connection")
00501         self._ser.close()
00502
00503
00504     """
00505 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00506 LLNL-CODE-838080
00507
00508 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00509 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00510 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00511 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00512 be made under this license.
00513 """

```

7.21 C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/Data.c File Reference

```

#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <memory.h>
#include "ZestETM1.h"
#include "Private.h"

```

Classes

- struct [ZESTETM1_WRITE_REG_CMD](#)
- struct [ZESTETM1_WRITE_REG_RESPONSE](#)
- struct [ZESTETM1_READ_REG_CMD](#)
- struct [ZESTETM1_READ_REG_RESPONSE](#)
- struct [ZESTETM1_MAILBOX_INT_CMD](#)
- struct [ZESTETM1_MAILBOX_INT_RESPONSE](#)

Macros

- #define [_CRT_SECURE_NO_WARNINGS](#)
- #define [ZESTETM1_COMMAND_SPI](#) 0xee
- #define [ZESTETM1_COMMAND_WRITE_REG](#) 0xf6
- #define [ZESTETM1_COMMAND_READ_REG](#) 0xf7
- #define [ZESTETM1_COMMAND_MAILBOX_INT](#) 0xf8

Functions

- [ZESTETM1_STATUS ZestETM1_OpenConnection](#) ([ZESTETM1_CARD_INFO](#) *CardInfo, [ZESTETM1_CONNECTION_TYPE](#) Type, [uint16_t](#) Port, [uint16_t](#) LocalPort, [ZESTETM1_CONNECTION](#) *Connection)
- [ZESTETM1_STATUS ZestETM1_CloseConnection](#) ([ZESTETM1_CONNECTION](#) Connection)
- static [ZESTETM1_STATUS ZestETM1_WriteData](#) ([ZESTETM1_CONNECTION](#) Connection, void *Buffer, [uint32_t](#) Length, unsigned long *Written, [uint32_t](#) Timeout)
- static [ZESTETM1_STATUS ZestETM1_ReadData](#) ([ZESTETM1_CONNECTION](#) Connection, void *Buffer, [uint32_t](#) Length, unsigned long *Read, [uint32_t](#) Timeout)
- [ZESTETM1_STATUS ZestETM1_SendCommand](#) ([ZESTETM1_CARD_INFO](#) *CardInfo, [ZESTETM1_CONNECTION](#) Connection, void *WriteBuffer, [uint32_t](#) WriteLen, void *ReadBuffer, [uint32_t](#) ReadLen, int WaitForAck)
- [ZESTETM1_STATUS ZestETM1_SPIReadWrite](#) ([ZESTETM1_CARD_INFO](#) *CardInfo, [ZESTETM1_CONNECTION](#) Connection, int Device, int WordLen, [uint32_t](#) *WriteData, [uint32_t](#) *ReadData, [uint32_t](#) Length, int ReleaseCS, int WaitForAck)
- [ZESTETM1_STATUS ZestETM1OpenConnection](#) ([ZESTETM1_CARD_INFO](#) *CardInfo, [ZESTETM1_CONNECTION_TYPE](#) Type, [uint16_t](#) Port, [uint16_t](#) LocalPort, [ZESTETM1_CONNECTION](#) *Connection)
- [ZESTETM1_STATUS ZestETM1CloseConnection](#) ([ZESTETM1_CONNECTION](#) Connection)
- [ZESTETM1_STATUS ZestETM1WriteData](#) ([ZESTETM1_CONNECTION](#) Connection, void *Buffer, unsigned long Length, unsigned long *Written, unsigned long Timeout)
- [ZESTETM1_STATUS ZestETM1ReadData](#) ([ZESTETM1_CONNECTION](#) Connection, void *Buffer, unsigned long Length, unsigned long *Read, unsigned long Timeout)
- [ZESTETM1_STATUS ZestETM1SPIReadWrite](#) ([ZESTETM1_CARD_INFO](#) *CardInfo, [ZESTETM1_SPI_RATE](#) Rate, int WordLen, void *WriteData, void *ReadData, unsigned long Length, int ReleaseCS)
- [ZESTETM1_STATUS ZestETM1WriteRegister](#) ([ZESTETM1_CARD_INFO](#) *CardInfo, unsigned long Addr, unsigned short Data)
- [ZESTETM1_STATUS ZestETM1ReadRegister](#) ([ZESTETM1_CARD_INFO](#) *CardInfo, unsigned long Addr, unsigned short *Data)
- [ZESTETM1_STATUS ZestETM1SetInterrupt](#) ([ZESTETM1_CARD_INFO](#) *CardInfo)

7.21.1 Class Documentation

7.21.1.1 struct ZESTETM1_WRITE_REG_CMD

Definition at line 58 of file [Data.c](#).

Class Members

uint8_t	Addr	
uint8_t	Command	
uint16_t	Data	

7.21.1.2 struct ZESTETM1_WRITE_REG_RESPONSE

Definition at line 64 of file [Data.c](#).

Class Members

uint8_t	Command	
-------------------------	---------	--

Class Members

uint8_t	Dummy1[2]	
uint8_t	Status	

7.21.1.3 struct ZESTETM1_READ_REG_CMD

Definition at line 70 of file [Data.c](#).

Class Members

uint8_t	Addr	
uint8_t	Command	
uint8_t	Dummy[2]	

7.21.1.4 struct ZESTETM1_READ_REG_RESPONSE

Definition at line 76 of file [Data.c](#).

Class Members

uint8_t	Command	
uint8_t	Status	
uint16_t	Value	

7.21.1.5 struct ZESTETM1_MAILBOX_INT_CMD

Definition at line 82 of file [Data.c](#).

Class Members

uint8_t	Command	
uint8_t	Dummy[3]	

7.21.1.6 struct ZESTETM1_MAILBOX_INT_RESPONSE

Definition at line 87 of file [Data.c](#).

Class Members

uint8_t	Command	
-------------------------	---------	--

Class Members

uint16_t	Dummy	
uint8_t	Status	

7.21.2 Macro Definition Documentation

7.21.2.1 _CRT_SECURE_NO_WARNINGS

```
#define _CRT_SECURE_NO_WARNINGS
```

Definition at line 45 of file [Data.c](#).

7.21.2.2 ZESTETM1_COMMAND_MAILBOX_INT

```
#define ZESTETM1_COMMAND_MAILBOX_INT 0xf8
```

Definition at line 96 of file [Data.c](#).

7.21.2.3 ZESTETM1_COMMAND_READ_REG

```
#define ZESTETM1_COMMAND_READ_REG 0xf7
```

Definition at line 95 of file [Data.c](#).

7.21.2.4 ZESTETM1_COMMAND_SPI

```
#define ZESTETM1_COMMAND_SPI 0xee
```

Definition at line 93 of file [Data.c](#).

7.21.2.5 ZESTETM1_COMMAND_WRITE_REG

```
#define ZESTETM1_COMMAND_WRITE_REG 0xf6
```

Definition at line 94 of file [Data.c](#).

7.21.3 Function Documentation

7.21.3.1 ZestETM1_CloseConnection()

```
ZESTETM1_STATUS ZestETM1_CloseConnection (
    ZESTETM1_CONNECTION Connection )
```

Definition at line 225 of file [Data.c](#).

```
00226 {
00227     ZESTETM1_CONNECTION_STRUCT *Conn = (ZESTETM1_CONNECTION_STRUCT *)Connection;
00228
00229     if (Conn==NULL)
00230     {
00231         return ZESTETM1_NULL_PARAMETER;
00232     }
00233     if (Conn->Magic!=ZESTETM1_CONNECTION_HANDLE_MAGIC)
00234     {
00235         return ZESTETM1_ILLEGAL_CONNECTION;
00236     }
00237
00238     // Cleanup
00239     closesocket (Conn->Socket);
00240     Conn->Magic = 0;
00241     free(Conn);
00242
00243     return ZESTETM1_SUCCESS;
00244 }
```

7.21.3.2 ZestETM1_OpenConnection()

```
ZESTETM1_STATUS ZestETM1_OpenConnection (
    ZESTETM1_CARD_INFO * CardInfo,
    ZESTETM1_CONNECTION_TYPE Type,
    uint16_t Port,
    uint16_t LocalPort,
    ZESTETM1_CONNECTION * Connection )
```

Definition at line 101 of file [Data.c](#).

```
00106 {
00107     ZESTETM1_CONNECTION_STRUCT *NewStruct;
00108     SOCKET Socket = -1;
00109     char AddrBuffer[32];
00110     char PortBuffer[32];
00111
00112     if (Connection==NULL || CardInfo==NULL)
00113     {
00114         return ZESTETM1_NULL_PARAMETER;
00115     }
00116
00117     // Allocate data structure
00118     NewStruct = malloc(sizeof(ZESTETM1_CONNECTION_STRUCT));
00119     if (NewStruct==NULL)
00120     {
00121         return ZESTETM1_OUT_OF_MEMORY;
00122     }
00123
00124     // Build target addresses
00125     sprintf(AddrBuffer, "%d.%d.%d.%d", CardInfo->IPAddr[0], CardInfo->IPAddr[1],
00126             CardInfo->IPAddr[2], CardInfo->IPAddr[3]);
00127     sprintf(PortBuffer, "%d", Port);
00128
00129     if (Type==ZESTETM1_TYPE_UDP)
00130     {
00131         // Open UDP connection
00132         struct sockaddr_in SourceIP;
00133         int SourceLen = (int)sizeof(struct sockaddr_in);
00134         Socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
00135         if (Socket<0)
```

```

00136         return ZESTETM1_SOCKET_ERROR;
00137     SourceIP.sin_family = AF_INET;
00138     SourceIP.sin_addr.s_addr = htonl(INADDR_ANY);
00139     SourceIP.sin_port = htons(LocalPort);
00140     bind(Socket, (const struct sockaddr *)&SourceIP, SourceLen);
00141
00142     NewStruct->Target.sin_family = AF_INET;
00143     NewStruct->Target.sin_addr.s_addr = inet_addr(AddrBuffer);
00144     NewStruct->Target.sin_port = htons(atoi(PortBuffer));
00145 }
00146 else if (Type==ZESTETM1_TYPE_TCP)
00147 {
00148     // Open TCP connection
00149     struct addrinfo *AddrResult = NULL,
00150                 *Ptr = NULL,
00151                 Hints;
00152     int Result;
00153     struct sockaddr_in SourceIP;
00154     int SourceLen = (int)sizeof(struct sockaddr_in);
00155
00156     memset(&Hints, 0, sizeof(Hints));
00157     Hints.ai_family = AF_UNSPEC;
00158     Hints.ai_socktype = SOCK_STREAM;
00159     Hints.ai_protocol = IPPROTO_TCP;
00160
00161     // Resolve the server address and port
00162     Result = getaddrinfo(AddrBuffer, PortBuffer, &Hints, &AddrResult);
00163     if (Result!=0)
00164     {
00165         free(NewStruct);
00166         return ZESTETM1_SOCKET_ERROR;
00167     }
00168
00169     // Attempt to connect to an address until one succeeds
00170     for (Ptr=AddrResult; Ptr!=NULL; Ptr=Ptr->ai_next)
00171     {
00172         // Create a SOCKET for connecting to server
00173         Socket = socket(Ptr->ai_family, Ptr->ai_socktype,
00174                       Ptr->ai_protocol);
00175         if (Socket<0)
00176         {
00177             freeaddrinfo(AddrResult);
00178             free(NewStruct);
00179             return ZESTETM1_SOCKET_ERROR;
00180         }
00181
00182         // Connect to ZestETM1
00183         Result = connect(Socket, Ptr->ai_addr, (int)Ptr->ai_addrlen);
00184         if (Result<0)
00185         {
00186             closesocket(Socket);
00187             Socket = -1;
00188             continue;
00189         }
00190         break;
00191     }
00192
00193     SourceIP.sin_family = AF_INET;
00194     SourceIP.sin_addr.s_addr = htonl(INADDR_ANY);
00195     SourceIP.sin_port = 0;
00196     bind(Socket, (const struct sockaddr *)&SourceIP, SourceLen);
00197     freeaddrinfo(AddrResult);
00198 }
00199 else
00200 {
00201     free(NewStruct);
00202     return ZESTETM1_INVALID_CONNECTION_TYPE;
00203 }
00204
00205 if (Socket==-1)
00206 {
00207     free(NewStruct);
00208     return ZESTETM1_SOCKET_ERROR;
00209 }
00210
00211 NewStruct->Magic = ZESTETM1_CONNECTION_HANDLE_MAGIC;
00212 NewStruct->Type = Type;
00213 NewStruct->Port = Port;
00214 NewStruct->LocalPort = LocalPort;
00215 NewStruct->Socket = Socket;
00216 NewStruct->CardInfo = CardInfo;

```

```

00217     *Connection = NewStruct;
00218
00219     return ZESTETM1_SUCCESS;
00220 }

```

7.21.3.3 ZestETM1_ReadData()

```

static ZESTETM1_STATUS ZestETM1_ReadData (
    ZESTETM1_CONNECTION Connection,
    void * Buffer,
    uint32_t Length,
    unsigned long * Read,
    uint32_t Timeout ) [static]

```

Definition at line 358 of file [Data.c](#).

```

00363 {
00364     ZESTETM1_CONNECTION_STRUCTURE *Conn = (ZESTETM1_CONNECTION_STRUCTURE *)Connection;
00365     fd_set ReadFDS;
00366     struct timeval Time;
00367     uint32_t BufferPtr = 0;
00368     int Result;
00369     struct sockaddr_in Target;
00370     int TargetLen = (int)sizeof(struct sockaddr_in);
00371
00372     if (Conn==NULL)
00373     {
00374         if (Read!=NULL) *Read = 0;
00375         return ZESTETM1_NULL_PARAMETER;
00376     }
00377     if (Conn->Magic!=ZESTETM1_CONNECTION_HANDLE_MAGIC)
00378     {
00379         if (Read!=NULL) *Read = 0;
00380         return ZESTETM1_ILLEGAL_CONNECTION;
00381     }
00382
00383     do
00384     {
00385         int OK = 1;
00386
00387         // Wait for socket to become ready
00388         {
00389             int CurTime = 0;
00390             while (CurTime!=Timeout)
00391             {
00392                 int T = (Timeout-CurTime)<1000 ? Timeout-CurTime : 1000;
00393                 FD_ZERO(&ReadFDS);
00394                 FD_SET(Conn->Socket, &ReadFDS);
00395                 Time.tv_sec = T/1000;
00396                 Time.tv_usec = (T%1000)*1000;
00397                 Result = select((int)Conn->Socket+1, &ReadFDS, NULL, NULL, &Time);
00398                 if (Result>0) break;
00399                 CurTime+=T;
00400             }
00401         }
00402         if (Result<0 || !FD_ISSET(Conn->Socket, &ReadFDS))
00403         {
00404             if (Read!=NULL) *Read = BufferPtr;
00405             return ZESTETM1_TIMEOUT;//FIXME: Any other errors?
00406         }
00407
00408         if (Conn->Type==ZESTETM1_TYPE_UDP)
00409         {
00410             Result = recvfrom(Conn->Socket, (char *)Buffer+BufferPtr,
00411                             Length-BufferPtr, 0,
00412                             (struct sockaddr *)&Target, &TargetLen);
00413             if (Conn->LocalPort!=0 && Target.sin_port!=htons(Conn->LocalPort))
00414                 OK = 0;
00415         }
00416         else if (Conn->Type==ZESTETM1_TYPE_TCP)
00417         {
00418             Result = recv(Conn->Socket, (char *)Buffer+BufferPtr,
00419                           Length-BufferPtr, 0);
00420         }

```

```

00421         else
00422         {
00423             if (Read!=NULL) *Read = BufferPtr;
00424             return ZESTETM1_ILLEGAL_CONNECTION;
00425         }
00426
00427         // Update counters
00428         if (Result>0 && OK==1)
00429         {
00430             BufferPtr += Result;
00431         }
00432         else if (Result==0)
00433         {
00434             // Connection closed
00435             if (Read!=NULL) *Read = BufferPtr;
00436             return ZESTETM1_SOCKET_CLOSED;
00437         }
00438         else if (Result<0)
00439         {
00440             // Socket error
00441             if (Read!=NULL) *Read = BufferPtr;
00442             return ZESTETM1_SOCKET_ERROR;
00443         }
00444     } while (Result>0 && BufferPtr<Length);
00445
00446     if (Read!=NULL) *Read = BufferPtr;
00447     return ZESTETM1_SUCCESS;
00448 }

```

7.21.3.4 ZestETM1_SendCommand()

```

ZESTETM1_STATUS ZestETM1_SendCommand (
    ZESTETM1_CARD_INFO * CardInfo,
    ZESTETM1_CONNECTION Connection,
    void * WriteBuffer,
    uint32_t WriteLen,
    void * ReadBuffer,
    uint32_t ReadLen,
    int WaitForAck )

```

Definition at line 453 of file [Data.c](#).

```

00458 {
00459     ZESTETM1_STATUS Result;
00460     unsigned long Written;
00461     unsigned long Received;
00462
00463     // Send/receive data
00464     Result = ZestETM1_WriteData(Connection, WriteBuffer, WriteLen, &Written,
00465                                CardInfo->Timeout);
00466     if (Result!=ZESTETM1_SUCCESS)
00467     {
00468         return Result;
00469     }
00470     if (Written!=WriteLen)
00471     {
00472         return ZESTETM1_INTERNAL_ERROR;
00473     }
00474     *((uint8_t *)ReadBuffer) = 0;
00475     if (WaitForAck==1)
00476     {
00477         Result = ZestETM1_ReadData(Connection, ReadBuffer, ReadLen, &Received,
00478                                   CardInfo->Timeout);
00479         if (Result!=ZESTETM1_SUCCESS)
00480         {
00481             return Result;
00482         }
00483         if (Received!=ReadLen)
00484         {
00485             return ZESTETM1_INTERNAL_ERROR;
00486         }
00487     }
00488     return ZESTETM1_SUCCESS;
00489 }
00490 }

```

7.21.3.5 ZestETM1_SPIReadWrite()

```

ZESTETM1_STATUS ZestETM1_SPIReadWrite (
    ZESTETM1_CARD_INFO * CardInfo,
    ZESTETM1_CONNECTION Connection,
    int Device,
    int WordLen,
    uint32_t * WriteData,
    uint32_t * ReadData,
    uint32_t Length,
    int ReleaseCS,
    int WaitForAck )

```

Definition at line 495 of file [Data.c](#).

```

00501 {
00502     uint8_t Buffer[65536];
00503     uint32_t *BufPtr;
00504     uint32_t i;
00505     ZESTETM1_STATUS Result;
00506
00507     // Build command
00508     Buffer[0] = ZESTETM1_COMMAND_SPI; // Command byte
00509     Buffer[1] = Device;                // SPI device
00510     Buffer[2] = WordLen;               // Word length
00511     Buffer[3] = ReleaseCS;             // Release CS on completion
00512     BufPtr = (uint32_t *) (Buffer+4);
00513
00514     BufPtr[0] = WriteData==NULL ? 0 : ZESTETM1_REVERSE(Length); // Num words
00515     BufPtr[1] = ReadData==NULL ? 0 : ZESTETM1_REVERSE(Length);
00516     if (WriteData!=NULL)
00517     {
00518         for (i=0; i<Length; i++)
00519         {
00520             BufPtr[2+i] = ZESTETM1_REVERSE(WriteData[i]);
00521         }
00522     }
00523     else
00524         memset(BufPtr+2, 0, Length*4);
00525
00526     // Send command
00527     Result = ZestETM1_SendCommand(CardInfo, Connection, Buffer,
00528                                   WriteData==NULL ? 12 : 12+Length*4,
00529                                   Buffer, ReadData==NULL ? 4 : 4+Length*4,
00530                                   WaitForAck);
00531     if (WaitForAck==0)
00532         return Result;
00533
00534     // Extract results
00535     if (Buffer[0]!=ZESTETM1_COMMAND_SPI || Buffer[1]!=0)
00536     {
00537         return ZESTETM1_INTERNAL_ERROR;
00538     }
00539
00540     BufPtr = (uint32_t *) (Buffer+4);
00541     if (ReadData!=NULL)
00542     {
00543         for (i=0; i<Length; i++)
00544         {
00545             ReadData[i] = ZESTETM1_REVERSE(BufPtr[i]);
00546         }
00547     }
00548
00549     return ZESTETM1_SUCCESS;
00550 }

```

7.21.3.6 ZestETM1_WriteData()

```

static ZESTETM1_STATUS ZestETM1_WriteData (
    ZESTETM1_CONNECTION Connection,

```

```

void * Buffer,
uint32_t Length,
unsigned long * Written,
uint32_t Timeout ) [static]

```

Definition at line 249 of file [Data.c](#).

```

00254 {
00255     ZESTETM1_CONNECTION_STRUCT *Conn = (ZESTETM1_CONNECTION_STRUCT *)Connection;
00256     fd_set WriteFDS;
00257     struct timeval Time;
00258     uint32_t BufferPtr = 0;
00259     int Result;
00260     int TargetLen = (int)sizeof(struct sockaddr_in);
00261     unsigned int MaxSize;
00262
00263     if (Conn==NULL)
00264     {
00265         if (Written!=NULL) *Written = 0;
00266         return ZESTETM1_NULL_PARAMETER;
00267     }
00268     if (Conn->Magic!=ZESTETM1_CONNECTION_HANDLE_MAGIC)
00269     {
00270         if (Written!=NULL) *Written = 0;
00271         return ZESTETM1_ILLEGAL_CONNECTION;
00272     }
00273
00274     if (Conn->Type==ZESTETM1_TYPE_UDP)
00275     {
00276         // Get maximum transfer size
00277 #if defined(MSVC) || defined(WINGCC)
00278         int MaxSizeLen = sizeof(MaxSize);
00279         getsockopt(Conn->Socket, SOL_SOCKET, SO_MAX_MSG_SIZE,
00280             (char *)&MaxSize, &MaxSizeLen);
00281 #else
00282         //FIXME: Linux doesn't support SO_MAX_MSG_SIZE
00283         MaxSize = 65507;
00284 #endif
00285     }
00286
00287     do
00288     {
00289         uint32_t Bytes = Length-BufferPtr;
00290
00291         if (Conn->Type==ZESTETM1_TYPE_UDP && Bytes>MaxSize)
00292             Bytes = MaxSize;
00293
00294         // Wait for socket to become ready
00295         {
00296             int CurTime = 0;
00297             while (CurTime!=Timeout)
00298             {
00299                 int T = (Timeout-CurTime)<1000 ? Timeout-CurTime : 1000;
00300                 FD_ZERO(&WriteFDS);
00301                 FD_SET(Conn->Socket, &WriteFDS);
00302                 Time.tv_sec = T/1000;
00303                 Time.tv_usec = (T%1000)*1000;
00304                 Result = select((int)Conn->Socket+1, NULL, &WriteFDS, NULL, &Time);
00305                 if (Result>0) break;
00306                 CurTime+=T;
00307             }
00308         }
00309         if (Result<0 || !FD_ISSET(Conn->Socket, &WriteFDS))
00310         {
00311             if (Written!=NULL) *Written = BufferPtr;
00312             return ZESTETM1_TIMEOUT;//FIXME: Any other errors?
00313         }
00314
00315         if (Conn->Type==ZESTETM1_TYPE_UDP)
00316         {
00317             Result = sendto(Conn->Socket, (char *)Buffer+BufferPtr,
00318                 Bytes, 0,
00319                 (struct sockaddr *)&Conn->Target, TargetLen);
00320         }
00321         else if (Conn->Type==ZESTETM1_TYPE_TCP)
00322         {
00323             Result = send(Conn->Socket, (char *)Buffer+BufferPtr,
00324                 Bytes, 0);
00325         }
00326         else

```

```

00327     {
00328         if (Written!=NULL) *Written = BufferPtr;
00329         return ZESTETM1_ILLEGAL_CONNECTION;
00330     }
00331
00332     // Update counters
00333     if (Result>0)
00334     {
00335         BufferPtr += Result;
00336     }
00337     else if (Result==0)
00338     {
00339         // Connection closed
00340         if (Written!=NULL) *Written = BufferPtr;
00341         return ZESTETM1_SOCKET_CLOSED;
00342     }
00343     else
00344     {
00345         // Socket error
00346         if (Written!=NULL) *Written = BufferPtr;
00347         return ZESTETM1_SOCKET_ERROR;
00348     }
00349 } while (Result>0 && BufferPtr<Length);
00350
00351 if (Written!=NULL) *Written = BufferPtr;
00352 return ZESTETM1_SUCCESS;
00353 }

```

7.21.3.7 ZestETM1CloseConnection()

```

ZESTETM1_STATUS ZestETM1CloseConnection (
    ZESTETM1_CONNECTION Connection )

```

Definition at line 575 of file [Data.c](#).

```

00576 {
00577     ZESTETM1_STATUS Result;
00578     ZESTETM1_CONNECTION_STRUCT *Conn = (ZESTETM1_CONNECTION_STRUCT*)Connection;
00579
00580     Result = ZestETM1_CloseConnection(Connection);
00581     if (Result!=ZESTETM1_SUCCESS)
00582     {
00583         ZESTETM1_ERROR_CONN("ZestETM1CloseConnection", Result);
00584     }
00585
00586     return ZESTETM1_SUCCESS;
00587 }

```

7.21.3.8 ZestETM1OpenConnection()

```

ZESTETM1_STATUS ZestETM1OpenConnection (
    ZESTETM1_CARD_INFO * CardInfo,
    ZESTETM1_CONNECTION_TYPE Type,
    uint16_t Port,
    uint16_t LocalPort,
    ZESTETM1_CONNECTION * Connection )

```

Definition at line 555 of file [Data.c](#).

```

00560 {
00561     ZESTETM1_STATUS Result;
00562
00563     Result = ZestETM1_OpenConnection(CardInfo, Type, Port, LocalPort, Connection);
00564     if (Result!=ZESTETM1_SUCCESS)
00565     {
00566         ZESTETM1_ERROR("ZestETM1OpenConnection", Result);
00567     }
00568
00569     return ZESTETM1_SUCCESS;
00570 }

```

7.21.3.9 ZestETM1ReadData()

```
ZESTETM1_STATUS ZestETM1ReadData (
    ZESTETM1_CONNECTION Connection,
    void * Buffer,
    unsigned long Length,
    unsigned long * Read,
    unsigned long Timeout )
```

Definition at line 613 of file [Data.c](#).

```
00618 {
00619     ZESTETM1_STATUS Result;
00620     ZESTETM1_CONNECTION_STRUCT *Conn = (ZESTETM1_CONNECTION_STRUCT*)Connection;
00621
00622     Result = ZestETM1_ReadData(Connection, Buffer, Length, Read, Timeout);
00623     if (Result!=ZESTETM1_SUCCESS)
00624     {
00625         ZESTETM1_ERROR_CONN("ZestETM1ReadData", Result);
00626     }
00627
00628     return ZESTETM1_SUCCESS;
00629 }
```

7.21.3.10 ZestETM1ReadRegister()

```
ZESTETM1_STATUS ZestETM1ReadRegister (
    ZESTETM1_CARD_INFO * CardInfo,
    unsigned long Addr,
    unsigned short * Data )
```

Definition at line 728 of file [Data.c](#).

```
00729 {
00730     ZESTETM1_STATUS Result;
00731     ZESTETM1_CONNECTION Connection;
00732     ZESTETM1_READ_REG_CMD Cmd;
00733     ZESTETM1_READ_REG_RESPONSE Response;
00734
00735     if (CardInfo==NULL || Data==NULL)
00736     {
00737         ZESTETM1_ERROR("ZestETM1ReadRegister", ZESTETM1_NULL_PARAMETER);
00738     }
00739     if (Addr>127)
00740     {
00741         ZESTETM1_ERROR("ZestETM1ReadRegister", ZESTETM1_ILLEGAL_PARAMETER);
00742     }
00743
00744     Result = ZestETM1_OpenConnection(CardInfo, ZESTETM1_TYPE_TCP, CardInfo->ControlPort, 0, &Connection);
00745     if (Result!=ZESTETM1_SUCCESS)
00746     {
00747         ZESTETM1_ERROR("ZestETM1ReadRegister", Result);
00748     }
00749
00750     // Read register from the device
00751     Cmd.Command = ZESTETM1_COMMAND_READ_REG;
00752     Cmd.Addr = (uint8_t)Addr;
00753     Result = ZestETM1_SendCommand(CardInfo, Connection,
00754                                   &Cmd, sizeof(Cmd),
00755                                   &Response, sizeof(Response), 1);
00756     if (Result!=ZESTETM1_SUCCESS)
00757     {
00758         ZestETM1_CloseConnection(Connection);
00759         ZESTETM1_ERROR("ZestETM1ReadRegister", Result);
00760     }
00761     if (Response.Command!=Cmd.Command || Response.Status!=0)
00762     {
00763         ZestETM1_CloseConnection(Connection);
00764         ZESTETM1_ERROR("ZestETM1ReadRegister", ZESTETM1_INTERNAL_ERROR);
00765     }
00766     *Data = ((Response.Value>>8)&0xff) | ((Response.Value&0xff)<<8);
```



```

00767
00768     Result = ZestETM1_CloseConnection(Connection);
00769     if (Result!=ZESTETM1_SUCCESS)
00770     {
00771         ZESTETM1_ERROR("ZestETM1ReadRegister", Result);
00772     }
00773
00774     return ZESTETM1_SUCCESS;
00775 }

```

7.21.3.11 ZestETM1SetInterrupt()

```

ZESTETM1_STATUS ZestETM1SetInterrupt (
    ZESTETM1_CARD_INFO * CardInfo )

```

Definition at line 780 of file [Data.c](#).

```

00781 {
00782     ZESTETM1_STATUS Result;
00783     ZESTETM1_CONNECTION Connection;
00784     ZESTETM1_MAILBOX_INT_CMD Cmd;
00785     ZESTETM1_MAILBOX_INT_RESPONSE Response;
00786
00787     if (CardInfo==NULL)
00788     {
00789         ZESTETM1_ERROR("ZestETM1SetInterrupt", ZESTETM1_NULL_PARAMETER);
00790     }
00791
00792     Result = ZestETM1_OpenConnection(CardInfo, ZESTETM1_TYPE_TCP, CardInfo->ControlPort, 0, &Connection);
00793     if (Result!=ZESTETM1_SUCCESS)
00794     {
00795         ZESTETM1_ERROR("ZestETM1SetInterrupt", Result);
00796     }
00797
00798     // Send command to set interrupt
00799     Cmd.Command = ZESTETM1_COMMAND_MAILBOX_INT;
00800     Result = ZestETM1_SendCommand(CardInfo, Connection,
00801                                 &Cmd, sizeof(Cmd),
00802                                 &Response, sizeof(Response), 1);
00803     if (Result!=ZESTETM1_SUCCESS)
00804     {
00805         ZestETM1_CloseConnection(Connection);
00806         ZESTETM1_ERROR("ZestETM1SetInterrupt", Result);
00807     }
00808     if (Response.Command!=Cmd.Command || Response.Status!=0)
00809     {
00810         ZestETM1_CloseConnection(Connection);
00811         ZESTETM1_ERROR("ZestETM1SetInterrupt", ZESTETM1_INTERNAL_ERROR);
00812     }
00813
00814     Result = ZestETM1_CloseConnection(Connection);
00815     if (Result!=ZESTETM1_SUCCESS)
00816     {
00817         ZESTETM1_ERROR("ZestETM1SetInterrupt", Result);
00818     }
00819
00820     return ZESTETM1_SUCCESS;
00821 }

```

7.21.3.12 ZestETM1SPIReadWrite()

```

ZESTETM1_STATUS ZestETM1SPIReadWrite (
    ZESTETM1_CARD_INFO * CardInfo,
    ZESTETM1_SPI_RATE Rate,
    int WordLen,
    void * WriteData,
    void * ReadData,

```

```
    unsigned long Length,
    int ReleaseCS )
```

Definition at line 635 of file [Data.c](#).

```
00639 {
00640     ZESTETM1_STATUS Result;
00641     ZESTETM1_CONNECTION Connection;
00642     unsigned long RateVal = Rate==ZESTETM1_SPI_RATE_35 ? ZESTETM1_RATE_40MHz :
00643         Rate==ZESTETM1_SPI_RATE_17_5 ? ZESTETM1_RATE_20MHz : ZESTETM1_RATE_10MHz;
00644
00645     if (CardInfo==NULL || (WriteData==NULL && ReadData==NULL))
00646     {
00647         ZESTETM1_ERROR("ZestETM1SPIReadWrite", ZESTETM1_NULL_PARAMETER);
00648     }
00649     if (WordLen<1 || WordLen>32 || Length>16384)
00650     {
00651         ZESTETM1_ERROR("ZestETM1SPIReadWrite", ZESTETM1_ILLEGAL_PARAMETER);
00652     }
00653
00654     Result = ZestETM1_OpenConnection(CardInfo, ZESTETM1_TYPE_TCP, CardInfo->ControlPort, 0, &Connection);
00655     if (Result!=ZESTETM1_SUCCESS)
00656     {
00657         ZESTETM1_ERROR("ZestETM1SPIReadWrite", Result);
00658     }
00659
00660     Result = ZestETM1_SPIReadWrite(CardInfo, Connection, ZESTETM1_USER_DEVICE_ID|RateVal, WordLen,
    (uint32_t *)WriteData, (uint32_t *)ReadData, Length, ReleaseCS, 1);
00661     if (Result!=ZESTETM1_SUCCESS)
00662     {
00663         ZESTETM1_ERROR("ZestETM1SPIReadWrite", Result);
00664     }
00665
00666     Result = ZestETM1_CloseConnection(Connection);
00667     if (Result!=ZESTETM1_SUCCESS)
00668     {
00669         ZESTETM1_ERROR("ZestETM1SPIReadWrite", Result);
00670     }
00671
00672     return ZESTETM1_SUCCESS;
00673 }
```

7.21.3.13 ZestETM1WriteData()

```
ZESTETM1_STATUS ZestETM1WriteData (
    ZESTETM1_CONNECTION Connection,
    void * Buffer,
    unsigned long Length,
    unsigned long * Written,
    unsigned long Timeout )
```

Definition at line 592 of file [Data.c](#).

```
00597 {
00598     ZESTETM1_STATUS Result;
00599     ZESTETM1_CONNECTION_STRUCT *Conn = (ZESTETM1_CONNECTION_STRUCT*)Connection;
00600
00601     Result = ZestETM1_WriteData(Connection, Buffer, Length, Written, Timeout);
00602     if (Result!=ZESTETM1_SUCCESS)
00603     {
00604         ZESTETM1_ERROR_CONN("ZestETM1WriteData", Result);
00605     }
00606
00607     return ZESTETM1_SUCCESS;
00608 }
```

7.21.3.14 ZestETM1WriteRegister()

```
ZESTETM1_STATUS ZestETM1WriteRegister (
    ZESTETM1_CARD_INFO * CardInfo,
```

```

    unsigned long Addr,
    unsigned short Data )

```

Definition at line 678 of file [Data.c](#).

```

00679 {
00680     ZESTETM1_STATUS Result;
00681     ZESTETM1_CONNECTION Connection;
00682     ZESTETM1_WRITE_REG_CMD Cmd;
00683     ZESTETM1_WRITE_REG_RESPONSE Response;
00684
00685     if (CardInfo==NULL)
00686     {
00687         ZESTETM1_ERROR("ZestETM1WriteRegister", ZESTETM1_NULL_PARAMETER);
00688     }
00689     if (Addr>127)
00690     {
00691         ZESTETM1_ERROR("ZestETM1WriteRegister", ZESTETM1_ILLEGAL_PARAMETER);
00692     }
00693
00694     Result = ZestETM1_OpenConnection(CardInfo, ZESTETM1_TYPE_TCP, CardInfo->ControlPort, 0, &Connection);
00695     if (Result!=ZESTETM1_SUCCESS)
00696     {
00697         ZESTETM1_ERROR("ZestETM1WriteRegister", Result);
00698     }
00699
00700     // Write register to the device
00701     //FIXME: Do we want to be able to copy this value to flash?
00702     Cmd.Command = ZESTETM1_COMMAND_WRITE_REG;
00703     Cmd.Addr = (uint8_t)Addr;
00704     Cmd.Data = ((Data>>8)&0xff) | (Data&0xff);
00705     Result = ZestETM1_SendCommand(CardInfo, Connection,
00706                                   &Cmd, sizeof(Cmd),
00707                                   &Response, sizeof(Response), 1);
00708     if (Result!=ZESTETM1_SUCCESS)
00709     {
00710         ZestETM1_CloseConnection(Connection);
00711         ZESTETM1_ERROR("ZestETM1WriteRegister", Result);
00712     }
00713     if (Response.Command!=Cmd.Command || Response.Status!=0)
00714     {
00715         ZestETM1_CloseConnection(Connection);
00716         ZESTETM1_ERROR("ZestETM1WriteRegister", ZESTETM1_INTERNAL_ERROR);
00717     }
00718
00719     Result = ZestETM1_CloseConnection(Connection);
00720     if (Result!=ZESTETM1_SUCCESS)
00721     {
00722         ZESTETM1_ERROR("ZestETM1WriteRegister", Result);
00723     }
00724
00725     return ZESTETM1_SUCCESS;
00726 }

```

7.22 Data.c

[Go to the documentation of this file.](#)

```

00001 // File:      Data.c
00002 //
00003 // Purpose:
00004 //     ZestETM1 Host Library
00005 //     Data transfer functions
00006 //
00007 // Version: 1.00
00008 // Date: 11/10/12
00009
00010 // Copyright (C) 2012 Orange Tree Technologies Ltd. All rights reserved.
00011 // Orange Tree Technologies grants the purchaser of a ZestETM1 the right to use and
00012 // modify this source code in any form in designs that target the ZestETM1.
00013 // Orange Tree Technologies prohibits the use of this source code or any modification of
00014 // it in any form in designs that target any other hardware unless the purchaser of the
00015 // ZestETM1 has purchased the appropriate licence from Orange Tree Technologies.
00016 // Contact Orange Tree Technologies if you want to purchase such a licence.
00017

```

```

00018 //*****
00019 /**
00020 /**  Disclaimer: LIMITED WARRANTY AND DISCLAIMER. These designs are
00021 /**      provided to you "as is". Orange Tree Technologies and its licensors
00022 /**      make and you receive no warranties or conditions, express, implied,
00023 /**      statutory or otherwise, and Orange Tree Technologies specifically
00024 /**      disclaims any implied warranties of merchantability, non-infringement,
00025 /**      or fitness for a particular purpose. Orange Tree Technologies does not
00026 /**      warrant that the functions contained in these designs will meet your
00027 /**      requirements, or that the operation of these designs will be
00028 /**      uninterrupted or error free, or that defects in the Designs will be
00029 /**      corrected. Furthermore, Orange Tree Technologies does not warrant or
00030 /**      make any representations regarding use or the results of the use of the
00031 /**      designs in terms of correctness, accuracy, reliability, or otherwise.
00032 /**
00033 /**      LIMITATION OF LIABILITY. In no event will Orange Tree Technologies
00034 /**      or its licensors be liable for any loss of data, lost profits, cost or
00035 /**      procurement of substitute goods or services, or for any special,
00036 /**      incidental, consequential, or indirect damages arising from the use or
00037 /**      operation of the designs or accompanying documentation, however caused
00038 /**      and on any theory of liability. This limitation will apply even if
00039 /**      Orange Tree Technologies has been advised of the possibility of such
00040 /**      damage. This limitation shall apply notwithstanding the failure of the
00041 /**      essential purpose of any limited remedies herein.
00042 /**
00043 //*****
00044
00045 #define _CRT_SECURE_NO_WARNINGS
00046 #ifdef WINGCC
00047 #define __USE_W32_SOCKETS
00048 #endif
00049
00050 #include <stdint.h>
00051 #include <stdlib.h>
00052 #include <stdio.h>
00053 #include <memory.h>
00054 #include "ZestETM1.h"
00055 #include "Private.h"
00056
00057 // Read/write register command structures
00058 typedef struct
00059 {
00060     uint8_t Command;
00061     uint8_t Addr;
00062     uint16_t Data;
00063 } ZESTETM1_WRITE_REG_CMD;
00064 typedef struct
00065 {
00066     uint8_t Command;
00067     uint8_t Status;
00068     uint8_t Dummy1[2];
00069 } ZESTETM1_WRITE_REG_RESPONSE;
00070 typedef struct
00071 {
00072     uint8_t Command;
00073     uint8_t Addr;
00074     uint8_t Dummy[2];
00075 } ZESTETM1_READ_REG_CMD;
00076 typedef struct
00077 {
00078     uint8_t Command;
00079     uint8_t Status;
00080     uint16_t Value;
00081 } ZESTETM1_READ_REG_RESPONSE;
00082 typedef struct
00083 {
00084     uint8_t Command;
00085     uint8_t Dummy[3];
00086 } ZESTETM1_MAILBOX_INT_CMD;
00087 typedef struct
00088 {
00089     uint8_t Command;
00090     uint8_t Status;
00091     uint16_t Dummy;
00092 } ZESTETM1_MAILBOX_INT_RESPONSE;
00093 #define ZESTETM1_COMMAND_SPI 0xee
00094 #define ZESTETM1_COMMAND_WRITE_REG 0xf6
00095 #define ZESTETM1_COMMAND_READ_REG 0xf7
00096 #define ZESTETM1_COMMAND_MAILBOX_INT 0xf8
00097
00098 //*****

```

```

00099 * Open a connection to a ZestETM1 for data transfer (internal version) *
00100 *****/
00101 ZESTETM1_STATUS ZestETM1_OpenConnection(ZESTETM1_CARD_INFO *CardInfo,
00102                                         ZESTETM1_CONNECTION_TYPE Type,
00103                                         uint16_t Port,
00104                                         uint16_t LocalPort,
00105                                         ZESTETM1_CONNECTION *Connection)
00106 {
00107     ZESTETM1_CONNECTION_STRUCT *NewStruct;
00108     SOCKET Socket = -1;
00109     char AddrBuffer[32];
00110     char PortBuffer[32];
00111
00112     if (Connection==NULL || CardInfo==NULL)
00113     {
00114         return ZESTETM1_NULL_PARAMETER;
00115     }
00116
00117     // Allocate data structure
00118     NewStruct = malloc(sizeof(ZESTETM1_CONNECTION_STRUCT));
00119     if (NewStruct==NULL)
00120     {
00121         return ZESTETM1_OUT_OF_MEMORY;
00122     }
00123
00124     // Build target addresses
00125     sprintf(AddrBuffer, "%d.%d.%d.%d", CardInfo->IPAddr[0], CardInfo->IPAddr[1],
00126           CardInfo->IPAddr[2], CardInfo->IPAddr[3]);
00127     sprintf(PortBuffer, "%d", Port);
00128
00129     if (Type==ZESTETM1_TYPE_UDP)
00130     {
00131         // Open UDP connection
00132         struct sockaddr_in SourceIP;
00133         int SourceLen = (int)sizeof(struct sockaddr_in);
00134         Socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
00135         if (Socket<0)
00136             return ZESTETM1_SOCKET_ERROR;
00137         SourceIP.sin_family = AF_INET;
00138         SourceIP.sin_addr.s_addr = htonl(INADDR_ANY);
00139         SourceIP.sin_port = htons(LocalPort);
00140         bind(Socket, (const struct sockaddr *)&SourceIP, SourceLen);
00141
00142         NewStruct->Target.sin_family = AF_INET;
00143         NewStruct->Target.sin_addr.s_addr = inet_addr(AddrBuffer);
00144         NewStruct->Target.sin_port = htons(atoi(PortBuffer));
00145     }
00146     else if (Type==ZESTETM1_TYPE_TCP)
00147     {
00148         // Open TCP connection
00149         struct addrinfo *AddrResult = NULL,
00150             *Ptr = NULL,
00151             Hints;
00152         int Result;
00153         struct sockaddr_in SourceIP;
00154         int SourceLen = (int)sizeof(struct sockaddr_in);
00155
00156         memset(&Hints, 0, sizeof(Hints));
00157         Hints.ai_family = AF_UNSPEC;
00158         Hints.ai_socktype = SOCK_STREAM;
00159         Hints.ai_protocol = IPPROTO_TCP;
00160
00161         // Resolve the server address and port
00162         Result = getaddrinfo(AddrBuffer, PortBuffer, &Hints, &AddrResult);
00163         if (Result!=0)
00164         {
00165             free(NewStruct);
00166             return ZESTETM1_SOCKET_ERROR;
00167         }
00168
00169         // Attempt to connect to an address until one succeeds
00170         for (Ptr=AddrResult; Ptr!=NULL; Ptr=Ptr->ai_next)
00171         {
00172             // Create a SOCKET for connecting to server
00173             Socket = socket(Ptr->ai_family, Ptr->ai_socktype,
00174                           Ptr->ai_protocol);
00175             if (Socket<0)
00176             {
00177                 freeaddrinfo(AddrResult);
00178                 free(NewStruct);
00179                 return ZESTETM1_SOCKET_ERROR;

```

```

00180         }
00181
00182         // Connect to ZestETM1
00183         Result = connect(Socket, Ptr->ai_addr, (int)Ptr->ai_addrlen);
00184         if (Result<0)
00185         {
00186             closesocket(Socket);
00187             Socket = -1;
00188             continue;
00189         }
00190         break;
00191     }
00192
00193     SourceIP.sin_family = AF_INET;
00194     SourceIP.sin_addr.s_addr = htonl(INADDR_ANY);
00195     SourceIP.sin_port = 0;
00196     bind(Socket, (const struct sockaddr *)&SourceIP, SourceLen);
00197     freeaddrinfo(AddrResult);
00198 }
00199 else
00200 {
00201     free(NewStruct);
00202     return ZESTETM1_INVALID_CONNECTION_TYPE;
00203 }
00204
00205 if (Socket==-1)
00206 {
00207     free(NewStruct);
00208     return ZESTETM1_SOCKET_ERROR;
00209 }
00210
00211 NewStruct->Magic = ZESTETM1_CONNECTION_HANDLE_MAGIC;
00212 NewStruct->Type = Type;
00213 NewStruct->Port = Port;
00214 NewStruct->LocalPort = LocalPort;
00215 NewStruct->Socket = Socket;
00216 NewStruct->CardInfo = CardInfo;
00217 *Connection = NewStruct;
00218
00219 return ZESTETM1_SUCCESS;
00220 }
00221
00222 /*****
00223 * Close a connection to a ZestETM1 (internal version) *
00224 *****/
00225 ZESTETM1_STATUS ZestETM1_CloseConnection(ZESTETM1_CONNECTION Connection)
00226 {
00227     ZESTETM1_CONNECTION_STRUCT *Conn = (ZESTETM1_CONNECTION_STRUCT *)Connection;
00228
00229     if (Conn==NULL)
00230     {
00231         return ZESTETM1_NULL_PARAMETER;
00232     }
00233     if (Conn->Magic!=ZESTETM1_CONNECTION_HANDLE_MAGIC)
00234     {
00235         return ZESTETM1_ILLEGAL_CONNECTION;
00236     }
00237
00238     // Cleanup
00239     closesocket(Conn->Socket);
00240     Conn->Magic = 0;
00241     free(Conn);
00242
00243     return ZESTETM1_SUCCESS;
00244 }
00245
00246 /*****
00247 * Write data to ZestETM1 connection (internal version) *
00248 *****/
00249 static ZESTETM1_STATUS ZestETM1_WriteData(ZESTETM1_CONNECTION Connection,
00250     void *Buffer,
00251     uint32_t Length,
00252     unsigned long *Written,
00253     uint32_t Timeout)
00254 {
00255     ZESTETM1_CONNECTION_STRUCT *Conn = (ZESTETM1_CONNECTION_STRUCT *)Connection;
00256     fd_set WriteFDS;
00257     struct timeval Time;
00258     uint32_t BufferPtr = 0;
00259     int Result;
00260     int TargetLen = (int)sizeof(struct sockaddr_in);

```

```

00261     unsigned int MaxSize;
00262
00263     if (Conn==NULL)
00264     {
00265         if (Written!=NULL) *Written = 0;
00266         return ZESTETM1_NULL_PARAMETER;
00267     }
00268     if (Conn->Magic!=ZESTETM1_CONNECTION_HANDLE_MAGIC)
00269     {
00270         if (Written!=NULL) *Written = 0;
00271         return ZESTETM1_ILLEGAL_CONNECTION;
00272     }
00273
00274     if (Conn->Type==ZESTETM1_TYPE_UDP)
00275     {
00276         // Get maximum transfer size
00277 #if defined(MSVVC) || defined(WINGCC)
00278         int MaxSizeLen = sizeof(MaxSize);
00279         getsockopt(Conn->Socket, SOL_SOCKET, SO_MAX_MSG_SIZE,
00280             (char *)&MaxSize, &MaxSizeLen);
00281 #else
00282         //FIXME: Linux doesn't support SO_MAX_MSG_SIZE
00283         MaxSize = 65507;
00284 #endif
00285     }
00286
00287     do
00288     {
00289         uint32_t Bytes = Length-BufferPtr;
00290
00291         if (Conn->Type==ZESTETM1_TYPE_UDP && Bytes>MaxSize)
00292             Bytes = MaxSize;
00293
00294         // Wait for socket to become ready
00295         {
00296             int CurTime = 0;
00297             while (CurTime!=Timeout)
00298             {
00299                 int T = (Timeout-CurTime)<1000 ? Timeout-CurTime : 1000;
00300                 FD_ZERO(&WriteFDS);
00301                 FD_SET(Conn->Socket, &WriteFDS);
00302                 Time.tv_sec = T/1000;
00303                 Time.tv_usec = (T%1000)*1000;
00304                 Result = select((int)Conn->Socket+1, NULL, &WriteFDS, NULL, &Time);
00305                 if (Result>0) break;
00306                 CurTime+=T;
00307             }
00308         }
00309         if (Result<0 || !FD_ISSET(Conn->Socket, &WriteFDS))
00310         {
00311             if (Written!=NULL) *Written = BufferPtr;
00312             return ZESTETM1_TIMEOUT;//FIXME: Any other errors?
00313         }
00314
00315         if (Conn->Type==ZESTETM1_TYPE_UDP)
00316         {
00317             Result = sendto(Conn->Socket, (char *)Buffer+BufferPtr,
00318                 Bytes, 0,
00319                 (struct sockaddr *)&Conn->Target, TargetLen);
00320         }
00321         else if (Conn->Type==ZESTETM1_TYPE_TCP)
00322         {
00323             Result = send(Conn->Socket, (char *)Buffer+BufferPtr,
00324                 Bytes, 0);
00325         }
00326         else
00327         {
00328             if (Written!=NULL) *Written = BufferPtr;
00329             return ZESTETM1_ILLEGAL_CONNECTION;
00330         }
00331
00332         // Update counters
00333         if (Result>0)
00334         {
00335             BufferPtr += Result;
00336         }
00337         else if (Result==0)
00338         {
00339             // Connection closed
00340             if (Written!=NULL) *Written = BufferPtr;
00341             return ZESTETM1_SOCKET_CLOSED;

```

```

00342     }
00343     else
00344     {
00345         // Socket error
00346         if (Written!=NULL) *Written = BufferPtr;
00347         return ZESTETM1_SOCKET_ERROR;
00348     }
00349 } while (Result>0 && BufferPtr<Length);
00350
00351 if (Written!=NULL) *Written = BufferPtr;
00352 return ZESTETM1_SUCCESS;
00353 }
00354
00355 /*****
00356 * Read data from a ZestETM1 connection (internal version) *
00357 *****/
00358 static ZESTETM1_STATUS ZestETM1_ReadData(ZESTETM1_CONNECTION Connection,
00359 void *Buffer,
00360 uint32_t Length,
00361 unsigned long *Read,
00362 uint32_t Timeout)
00363 {
00364     ZESTETM1_CONNECTION_STRUCT *Conn = (ZESTETM1_CONNECTION_STRUCT *)Connection;
00365     fd_set ReadFDS;
00366     struct timeval Time;
00367     uint32_t BufferPtr = 0;
00368     int Result;
00369     struct sockaddr_in Target;
00370     int TargetLen = (int)sizeof(struct sockaddr_in);
00371
00372     if (Conn==NULL)
00373     {
00374         if (Read!=NULL) *Read = 0;
00375         return ZESTETM1_NULL_PARAMETER;
00376     }
00377     if (Conn->Magic!=ZESTETM1_CONNECTION_HANDLE_MAGIC)
00378     {
00379         if (Read!=NULL) *Read = 0;
00380         return ZESTETM1_ILLEGAL_CONNECTION;
00381     }
00382
00383     do
00384     {
00385         int OK = 1;
00386
00387         // Wait for socket to become ready
00388         {
00389             int CurTime = 0;
00390             while (CurTime!=Timeout)
00391             {
00392                 int T = (Timeout-CurTime)<1000 ? Timeout-CurTime : 1000;
00393                 FD_ZERO(&ReadFDS);
00394                 FD_SET(Conn->Socket, &ReadFDS);
00395                 Time.tv_sec = T/1000;
00396                 Time.tv_usec = (T%1000)*1000;
00397                 Result = select((int)Conn->Socket+1, &ReadFDS, NULL, NULL, &Time);
00398                 if (Result>0) break;
00399                 CurTime+=T;
00400             }
00401         }
00402         if (Result<0 || !FD_ISSET(Conn->Socket, &ReadFDS))
00403         {
00404             if (Read!=NULL) *Read = BufferPtr;
00405             return ZESTETM1_TIMEOUT;//FIXME: Any other errors?
00406         }
00407
00408         if (Conn->Type==ZESTETM1_TYPE_UDP)
00409         {
00410             Result = recvfrom(Conn->Socket, (char *)Buffer+BufferPtr,
00411                             Length-BufferPtr, 0,
00412                             (struct sockaddr *)&Target, &TargetLen);
00413             if (Conn->LocalPort!=0 && Target.sin_port!=htons(Conn->LocalPort))
00414                 OK = 0;
00415         }
00416         else if (Conn->Type==ZESTETM1_TYPE_TCP)
00417         {
00418             Result = recv(Conn->Socket, (char *)Buffer+BufferPtr,
00419                           Length-BufferPtr, 0);
00420         }
00421         else
00422         {

```



```

00423         if (Read!=NULL) *Read = BufferPtr;
00424         return ZESTETM1_ILLEGAL_CONNECTION;
00425     }
00426
00427     // Update counters
00428     if (Result>0 && OK==1)
00429     {
00430         BufferPtr += Result;
00431     }
00432     else if (Result==0)
00433     {
00434         // Connection closed
00435         if (Read!=NULL) *Read = BufferPtr;
00436         return ZESTETM1_SOCKET_CLOSED;
00437     }
00438     else if (Result<0)
00439     {
00440         // Socket error
00441         if (Read!=NULL) *Read = BufferPtr;
00442         return ZESTETM1_SOCKET_ERROR;
00443     }
00444     } while (Result>0 && BufferPtr<Length);
00445
00446     if (Read!=NULL) *Read = BufferPtr;
00447     return ZESTETM1_SUCCESS;
00448 }
00449
00450 /*****
00451 * Send a control command to GigEx and get response *
00452 *****/
00453 ZESTETM1_STATUS ZestETM1_SendCommand(ZESTETM1_CARD_INFO *CardInfo,
00454                                     ZESTETM1_CONNECTION Connection,
00455                                     void *WriteBuffer, uint32_t WriteLen,
00456                                     void *ReadBuffer, uint32_t ReadLen,
00457                                     int WaitForAck)
00458 {
00459     ZESTETM1_STATUS Result;
00460     unsigned long Written;
00461     unsigned long Received;
00462
00463     // Send/receive data
00464     Result = ZestETM1_WriteData(Connection, WriteBuffer, WriteLen, &Written,
00465                                CardInfo->Timeout);
00466     if (Result!=ZESTETM1_SUCCESS)
00467     {
00468         return Result;
00469     }
00470     if (Written!=WriteLen)
00471     {
00472         return ZESTETM1_INTERNAL_ERROR;
00473     }
00474     *((uint8_t *)ReadBuffer) = 0;
00475     if (WaitForAck==1)
00476     {
00477         Result = ZestETM1_ReadData(Connection, ReadBuffer, ReadLen, &Received,
00478                                   CardInfo->Timeout);
00479         if (Result!=ZESTETM1_SUCCESS)
00480         {
00481             return Result;
00482         }
00483         if (Received!=ReadLen)
00484         {
00485             return ZESTETM1_INTERNAL_ERROR;
00486         }
00487     }
00488
00489     return ZESTETM1_SUCCESS;
00490 }
00491
00492 /*****
00493 * Send data to/from SPI port on GigEx device *
00494 *****/
00495 ZESTETM1_STATUS ZestETM1_SPIReadWrite(ZESTETM1_CARD_INFO *CardInfo,
00496                                       ZESTETM1_CONNECTION Connection,
00497                                       int Device,
00498                                       int WordLen, uint32_t *WriteData,
00499                                       uint32_t *ReadData, uint32_t Length,
00500                                       int ReleaseCS, int WaitForAck)
00501 {
00502     uint8_t Buffer[65536];
00503     uint32_t *BufPtr;

```

```

00504     uint32_t i;
00505     ZESTETM1_STATUS Result;
00506
00507     // Build command
00508     Buffer[0] = ZESTETM1_COMMAND_SPI; // Command byte
00509     Buffer[1] = Device;                // SPI device
00510     Buffer[2] = WordLen;                // Word length
00511     Buffer[3] = ReleaseCS;              // Release CS on completion
00512     BufPtr = (uint32_t *) (Buffer+4);
00513
00514     BufPtr[0] = WriteData==NULL ? 0 : ZESTETM1_REVERSE(Length); // Num words
00515     BufPtr[1] = ReadData==NULL ? 0 : ZESTETM1_REVERSE(Length);
00516     if (WriteData!=NULL)
00517     {
00518         for (i=0; i<Length; i++)
00519         {
00520             BufPtr[2+i] = ZESTETM1_REVERSE(WriteData[i]);
00521         }
00522     }
00523     else
00524         memset(BufPtr+2, 0, Length*4);
00525
00526     // Send command
00527     Result = ZestETM1_SendCommand(CardInfo, Connection, Buffer,
00528                                   WriteData==NULL ? 12 : 12+Length*4,
00529                                   Buffer, ReadData==NULL ? 4 : 4+Length*4,
00530                                   WaitForAck);
00531     if (WaitForAck==0)
00532         return Result;
00533
00534     // Extract results
00535     if (Buffer[0]!=ZESTETM1_COMMAND_SPI || Buffer[1]!=0)
00536     {
00537         return ZESTETM1_INTERNAL_ERROR;
00538     }
00539
00540     BufPtr = (uint32_t *) (Buffer+4);
00541     if (ReadData!=NULL)
00542     {
00543         for (i=0; i<Length; i++)
00544         {
00545             ReadData[i] = ZESTETM1_REVERSE(BufPtr[i]);
00546         }
00547     }
00548
00549     return ZESTETM1_SUCCESS;
00550 }
00551
00552 /*****
00553 * Open a connection to a ZestETM1 for data transfer *
00554 *****/
00555 ZESTETM1_STATUS ZestETM1OpenConnection(ZESTETM1_CARD_INFO *CardInfo,
00556                                         ZESTETM1_CONNECTION_TYPE Type,
00557                                         uint16_t Port,
00558                                         uint16_t LocalPort,
00559                                         ZESTETM1_CONNECTION *Connection)
00560 {
00561     ZESTETM1_STATUS Result;
00562
00563     Result = ZestETM1_OpenConnection(CardInfo, Type, Port, LocalPort, Connection);
00564     if (Result!=ZESTETM1_SUCCESS)
00565     {
00566         ZESTETM1_ERROR("ZestETM1OpenConnection", Result);
00567     }
00568
00569     return ZESTETM1_SUCCESS;
00570 }
00571
00572 /*****
00573 * Close a connection to a ZestETM1 *
00574 *****/
00575 ZESTETM1_STATUS ZestETM1CloseConnection(ZESTETM1_CONNECTION Connection)
00576 {
00577     ZESTETM1_STATUS Result;
00578     ZESTETM1_CONNECTION_STRUCT *Conn = (ZESTETM1_CONNECTION_STRUCT*)Connection;
00579
00580     Result = ZestETM1_CloseConnection(Connection);
00581     if (Result!=ZESTETM1_SUCCESS)
00582     {
00583         ZESTETM1_ERROR_CONN("ZestETM1CloseConnection", Result);
00584     }

```

```

00585
00586     return ZESTETM1_SUCCESS;
00587 }
00588
00589 /*****
00590  * Write data to ZestETM1 connection *
00591  *****/
00592 ZESTETM1_STATUS ZestETM1WriteData(ZESTETM1_CONNECTION Connection,
00593     void *Buffer,
00594     unsigned long Length,
00595     unsigned long *Written,
00596     unsigned long Timeout)
00597 {
00598     ZESTETM1_STATUS Result;
00599     ZESTETM1_CONNECTION_STRUCT *Conn = (ZESTETM1_CONNECTION_STRUCT*)Connection;
00600
00601     Result = ZestETM1_WriteData(Connection, Buffer, Length, Written, Timeout);
00602     if (Result!=ZESTETM1_SUCCESS)
00603     {
00604         ZESTETM1_ERROR_CONN("ZestETM1WriteData", Result);
00605     }
00606
00607     return ZESTETM1_SUCCESS;
00608 }
00609
00610 /*****
00611  * Read data from a ZestETM1 connection *
00612  *****/
00613 ZESTETM1_STATUS ZestETM1ReadData(ZESTETM1_CONNECTION Connection,
00614     void *Buffer,
00615     unsigned long Length,
00616     unsigned long *Read,
00617     unsigned long Timeout)
00618 {
00619     ZESTETM1_STATUS Result;
00620     ZESTETM1_CONNECTION_STRUCT *Conn = (ZESTETM1_CONNECTION_STRUCT*)Connection;
00621
00622     Result = ZestETM1_ReadData(Connection, Buffer, Length, Read, Timeout);
00623     if (Result!=ZESTETM1_SUCCESS)
00624     {
00625         ZESTETM1_ERROR_CONN("ZestETM1ReadData", Result);
00626     }
00627
00628     return ZESTETM1_SUCCESS;
00629 }
00630
00631
00632 /*****
00633  * Read data from master SPI port *
00634  *****/
00635 ZESTETM1_STATUS ZestETM1SPIReadWrite(ZESTETM1_CARD_INFO *CardInfo, ZESTETM1_SPI_RATE Rate,
00636     int WordLen, void *WriteData,
00637     void *ReadData, unsigned long Length,
00638     int ReleaseCS)
00639 {
00640     ZESTETM1_STATUS Result;
00641     ZESTETM1_CONNECTION Connection;
00642     unsigned long RateVal = Rate==ZESTETM1_SPI_RATE_35 ? ZESTETM1_RATE_40MHZ :
00643         Rate==ZESTETM1_SPI_RATE_17_5 ? ZESTETM1_RATE_20MHZ : ZESTETM1_RATE_10MHZ;
00644
00645     if (CardInfo==NULL || (WriteData==NULL && ReadData==NULL))
00646     {
00647         ZESTETM1_ERROR("ZestETM1SPIReadWrite", ZESTETM1_NULL_PARAMETER);
00648     }
00649     if (WordLen<1 || WordLen>32 || Length>16384)
00650     {
00651         ZESTETM1_ERROR("ZestETM1SPIReadWrite", ZESTETM1_ILLEGAL_PARAMETER);
00652     }
00653
00654     Result = ZestETM1_OpenConnection(CardInfo, ZESTETM1_TYPE_TCP, CardInfo->ControlPort, 0, &Connection);
00655     if (Result!=ZESTETM1_SUCCESS)
00656     {
00657         ZESTETM1_ERROR("ZestETM1SPIReadWrite", Result);
00658     }
00659
00660     Result = ZestETM1_SPIReadWrite(CardInfo, Connection, ZESTETM1_USER_DEVICE_ID|RateVal, WordLen,
00661     (uint32_t *)WriteData, (uint32_t *)ReadData, Length, ReleaseCS, 1);
00662     if (Result!=ZESTETM1_SUCCESS)
00663     {
00664         ZESTETM1_ERROR("ZestETM1SPIReadWrite", Result);
00665     }
00666 }

```

```

00665
00666     Result = ZestETM1_CloseConnection(Connection);
00667     if (Result!=ZESTETM1_SUCCESS)
00668     {
00669         ZESTETM1_ERROR("ZestETM1SPIReadWrite", Result);
00670     }
00671
00672     return ZESTETM1_SUCCESS;
00673 }
00674
00675 /*****
00676  * Read/Write user interface registers *
00677  *****/
00678 ZESTETM1_STATUS ZestETM1WriteRegister(ZESTETM1_CARD_INFO *CardInfo, unsigned long Addr, unsigned short
Data)
00679 {
00680     ZESTETM1_STATUS Result;
00681     ZESTETM1_CONNECTION Connection;
00682     ZESTETM1_WRITE_REG_CMD Cmd;
00683     ZESTETM1_WRITE_REG_RESPONSE Response;
00684
00685     if (CardInfo==NULL)
00686     {
00687         ZESTETM1_ERROR("ZestETM1WriteRegister", ZESTETM1_NULL_PARAMETER);
00688     }
00689     if (Addr>127)
00690     {
00691         ZESTETM1_ERROR("ZestETM1WriteRegister", ZESTETM1_ILLEGAL_PARAMETER);
00692     }
00693
00694     Result = ZestETM1_OpenConnection(CardInfo, ZESTETM1_TYPE_TCP, CardInfo->ControlPort, 0, &Connection);
00695     if (Result!=ZESTETM1_SUCCESS)
00696     {
00697         ZESTETM1_ERROR("ZestETM1WriteRegister", Result);
00698     }
00699
00700     // Write register to the device
00701     //FIXME: Do we want to be able to copy this value to flash?
00702     Cmd.Command = ZESTETM1_COMMAND_WRITE_REG;
00703     Cmd.Addr = (uint8_t)Addr;
00704     Cmd.Data = ((Data>8)&0xff) | (Data<8?0:0xff);
00705     Result = ZestETM1_SendCommand(CardInfo, Connection,
                                &Cmd, sizeof(Cmd),
                                &Response, sizeof(Response), 1);
00706
00707     if (Result!=ZESTETM1_SUCCESS)
00708     {
00709         ZestETM1_CloseConnection(Connection);
00710         ZESTETM1_ERROR("ZestETM1WriteRegister", Result);
00711     }
00712     if (Response.Command!=Cmd.Command || Response.Status!=0)
00713     {
00714         ZestETM1_CloseConnection(Connection);
00715         ZESTETM1_ERROR("ZestETM1WriteRegister", ZESTETM1_INTERNAL_ERROR);
00716     }
00717
00718     Result = ZestETM1_CloseConnection(Connection);
00719     if (Result!=ZESTETM1_SUCCESS)
00720     {
00721         ZESTETM1_ERROR("ZestETM1WriteRegister", Result);
00722     }
00723
00724     return ZESTETM1_SUCCESS;
00725 }
00726
00727
00728 ZESTETM1_STATUS ZestETM1ReadRegister(ZESTETM1_CARD_INFO *CardInfo, unsigned long Addr, unsigned short
*Data)
00729 {
00730     ZESTETM1_STATUS Result;
00731     ZESTETM1_CONNECTION Connection;
00732     ZESTETM1_READ_REG_CMD Cmd;
00733     ZESTETM1_READ_REG_RESPONSE Response;
00734
00735     if (CardInfo==NULL || Data==NULL)
00736     {
00737         ZESTETM1_ERROR("ZestETM1ReadRegister", ZESTETM1_NULL_PARAMETER);
00738     }
00739     if (Addr>127)
00740     {
00741         ZESTETM1_ERROR("ZestETM1ReadRegister", ZESTETM1_ILLEGAL_PARAMETER);
00742     }
00743

```

```

00744     Result = ZestETM1_OpenConnection(CardInfo, ZESTETM1_TYPE_TCP, CardInfo->ControlPort, 0, &Connection);
00745     if (Result!=ZESTETM1_SUCCESS)
00746     {
00747         ZESTETM1_ERROR("ZestETM1ReadRegister", Result);
00748     }
00749
00750     // Read register from the device
00751     Cmd.Command = ZESTETM1_COMMAND_READ_REG;
00752     Cmd.Addr = (uint8_t)Addr;
00753     Result = ZestETM1_SendCommand(CardInfo, Connection,
00754                                   &Cmd, sizeof(Cmd),
00755                                   &Response, sizeof(Response), 1);
00756     if (Result!=ZESTETM1_SUCCESS)
00757     {
00758         ZestETM1_CloseConnection(Connection);
00759         ZESTETM1_ERROR("ZestETM1ReadRegister", Result);
00760     }
00761     if (Response.Command!=Cmd.Command || Response.Status!=0)
00762     {
00763         ZestETM1_CloseConnection(Connection);
00764         ZESTETM1_ERROR("ZestETM1ReadRegister", ZESTETM1_INTERNAL_ERROR);
00765     }
00766     *Data = ((Response.Value>>8)&0xff) | ((Response.Value&0xff)<<8);
00767
00768     Result = ZestETM1_CloseConnection(Connection);
00769     if (Result!=ZESTETM1_SUCCESS)
00770     {
00771         ZESTETM1_ERROR("ZestETM1ReadRegister", Result);
00772     }
00773
00774     return ZESTETM1_SUCCESS;
00775 }
00776
00777 /*****
00778 * Set mailbox interrupt *
00779 *****/
00780 ZESTETM1_STATUS ZestETM1SetInterrupt(ZESTETM1_CARD_INFO *CardInfo)
00781 {
00782     ZESTETM1_STATUS Result;
00783     ZESTETM1_CONNECTION Connection;
00784     ZESTETM1_MAILBOX_INT_CMD Cmd;
00785     ZESTETM1_MAILBOX_INT_RESPONSE Response;
00786
00787     if (CardInfo==NULL)
00788     {
00789         ZESTETM1_ERROR("ZestETM1SetInterrupt", ZESTETM1_NULL_PARAMETER);
00790     }
00791
00792     Result = ZestETM1_OpenConnection(CardInfo, ZESTETM1_TYPE_TCP, CardInfo->ControlPort, 0, &Connection);
00793     if (Result!=ZESTETM1_SUCCESS)
00794     {
00795         ZESTETM1_ERROR("ZestETM1SetInterrupt", Result);
00796     }
00797
00798     // Send command to set interrupt
00799     Cmd.Command = ZESTETM1_COMMAND_MAILBOX_INT;
00800     Result = ZestETM1_SendCommand(CardInfo, Connection,
00801                                   &Cmd, sizeof(Cmd),
00802                                   &Response, sizeof(Response), 1);
00803     if (Result!=ZESTETM1_SUCCESS)
00804     {
00805         ZestETM1_CloseConnection(Connection);
00806         ZESTETM1_ERROR("ZestETM1SetInterrupt", Result);
00807     }
00808     if (Response.Command!=Cmd.Command || Response.Status!=0)
00809     {
00810         ZestETM1_CloseConnection(Connection);
00811         ZESTETM1_ERROR("ZestETM1SetInterrupt", ZESTETM1_INTERNAL_ERROR);
00812     }
00813
00814     Result = ZestETM1_CloseConnection(Connection);
00815     if (Result!=ZESTETM1_SUCCESS)
00816     {
00817         ZESTETM1_ERROR("ZestETM1SetInterrupt", Result);
00818     }
00819
00820     return ZESTETM1_SUCCESS;
00821 }

```

7.23 C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/Error.c

File Reference

```
#include <stdint.h>
#include "ZestETM1.h"
#include "Private.h"
```

Functions

- [ZESTETM1_STATUS ZestETM1RegisterErrorHandler](#) ([ZESTETM1_ERROR_FUNC](#) Function)
- [ZESTETM1_STATUS ZestETM1GetErrorMessage](#) ([ZESTETM1_STATUS](#) Status, char **Buffer)

Variables

- char * [ZestETM1_ErrorStrings](#) []
- [ZESTETM1_ERROR_FUNC ZestETM1_ErrorHandler](#)

7.23.1 Function Documentation

7.23.1.1 ZestETM1GetErrorMessage()

```
ZESTETM1_STATUS ZestETM1GetErrorMessage (
    ZESTETM1_STATUS Status,
    char ** Buffer )
```

Definition at line 84 of file [Error.c](#).

```
00086 {
00087     if (Status>ZESTETM1_MAX_ERROR ||
00088         (Status<ZESTETM1_ERROR_BASE && Status>=ZESTETM1_MAX_WARNING) ||
00089         (Status<ZESTETM1_WARNING_BASE && Status>=ZESTETM1_MAX_INFO))
00090     {
00091         return ZESTETM1_ILLEGAL_STATUS_CODE;
00092     }
00093     *Buffer = ZESTETM1_ERROR_STRING(Status);
00094     return ZESTETM1_SUCCESS;
00095 }
00096 }
```

7.23.1.2 ZestETM1RegisterErrorHandler()

```
ZESTETM1_STATUS ZestETM1RegisterErrorHandler (
    ZESTETM1_ERROR_FUNC Function )
```

Definition at line 74 of file [Error.c](#).

```
00075 {
00076     ZestETM1_ErrorHandler = Function;
00077     return ZESTETM1_SUCCESS;
00078 }
```

7.23.2 Variable Documentation

7.23.2.1 ZestETM1_ErrorHandler

[ZESTETM1_ERROR_FUNC](#) ZestETM1_ErrorHandler

Definition at line 67 of file [Error.c](#).

7.23.2.2 ZestETM1_ErrorStrings

```
char* ZestETM1_ErrorStrings[]
```

Initial value:

```
=
{
    "Success (no error)",
    "Error communicating with socket",
    "An unspecified internal error occurred",
    "Status code is out of range",
    "NULL was used illegally as one of the parameter values",
    "Not enough memory to complete the requested operation",
    "The requested connection type is invalid",
    "The requested connection is invalid",
    "The connection was closed unexpectedly",
    "Operation timed out",
    "One of the parameters has an illegal value",
}
```

Definition at line 52 of file [Error.c](#).

```
00053 {
00054     "Success (no error)",
00055     "Error communicating with socket",
00056     "An unspecified internal error occurred",
00057     "Status code is out of range",
00058     "NULL was used illegally as one of the parameter values",
00059     "Not enough memory to complete the requested operation",
00060     "The requested connection type is invalid",
00061     "The requested connection is invalid",
00062     "The connection was closed unexpectedly",
00063     "Operation timed out",
00064     "One of the parameters has an illegal value",
00065
00066 };
```

7.24 Error.c

[Go to the documentation of this file.](#)

```
00001 // File:      Error.c
00002 //
00003 // Purpose:
00004 //     ZestETM1 Host Library
00005 //     Error functions
00006 //
00007 // Version: 1.00
00008 // Date: 11/10/12
00009
00010 // Copyright (C) 2012 Orange Tree Technologies Ltd. All rights reserved.
00011 // Orange Tree Technologies grants the purchaser of a ZestETM1 the right to use and
00012 // modify this source code in any form in designs that target the ZestETM1.
00013 // Orange Tree Technologies prohibits the use of this source code or any modification of
00014 // it in any form in designs that target any other hardware unless the purchaser of the
00015 // ZestETM1 has purchased the appropriate licence from Orange Tree Technologies.
00016 // Contact Orange Tree Technologies if you want to purchase such a licence.
```

```

00017
00018 //*****
00019 /**
00020 /** Disclaimer: LIMITED WARRANTY AND DISCLAIMER. These designs are
00021 /** provided to you "as is". Orange Tree Technologies and its licensors
00022 /** make and you receive no warranties or conditions, express, implied,
00023 /** statutory or otherwise, and Orange Tree Technologies specifically
00024 /** disclaims any implied warranties of merchantability, non-infringement,
00025 /** or fitness for a particular purpose. Orange Tree Technologies does not
00026 /** warrant that the functions contained in these designs will meet your
00027 /** requirements, or that the operation of these designs will be
00028 /** uninterrupted or error free, or that defects in the Designs will be
00029 /** corrected. Furthermore, Orange Tree Technologies does not warrant or
00030 /** make any representations regarding use or the results of the use of the
00031 /** designs in terms of correctness, accuracy, reliability, or otherwise.
00032 /**
00033 /** LIMITATION OF LIABILITY. In no event will Orange Tree Technologies
00034 /** or its licensors be liable for any loss of data, lost profits, cost or
00035 /** procurement of substitute goods or services, or for any special,
00036 /** incidental, consequential, or indirect damages arising from the use or
00037 /** operation of the designs or accompanying documentation, however caused
00038 /** and on any theory of liability. This limitation will apply even if
00039 /** Orange Tree Technologies has been advised of the possibility of such
00040 /** damage. This limitation shall apply notwithstanding the failure of the
00041 /** essential purpose of any limited remedies herein.
00042 /**
00043 //*****
00044
00045 #include <stdint.h>
00046 #include "ZestETM1.h"
00047 #include "Private.h"
00048
00049 /*****
00050 * Globals
00051 *****/
00052 char *ZestETM1_ErrorStrings[] =
00053 {
00054     "Success (no error)",
00055     "Error communicating with socket",
00056     "An unspecified internal error occurred",
00057     "Status code is out of range",
00058     "NULL was used illegally as one of the parameter values",
00059     "Not enough memory to complete the requested operation",
00060     "The requested connection type is invalid",
00061     "The requested connection is invalid",
00062     "The connection was closed unexpectedly",
00063     "Operation timed out",
00064     "One of the parameters has an illegal value",
00065 };
00066 };
00067 ZESTETM1_ERROR_FUNC ZestETM1_ErrorHandler;
00068
00069 /*****
00070 * Register a user error handling function to be called
00071 * Set to NULL to disable error callbacks
00072 *****/
00073 ZESTETM1_STATUS ZestETM1RegisterErrorHandler(ZESTETM1_ERROR_FUNC Function)
00074 {
00075     ZestETM1_ErrorHandler = Function;
00076     return ZESTETM1_SUCCESS;
00077 }
00078
00079 /*****
00080 * Get a human-readable error string for a status code
00081 *****/
00082 ZESTETM1_STATUS ZestETM1GetErrorMessage(ZESTETM1_STATUS Status,
00083     char **Buffer)
00084 {
00085     if (Status > ZESTETM1_MAX_ERROR ||
00086         (Status < ZESTETM1_ERROR_BASE && Status >= ZESTETM1_MAX_WARNING) ||
00087         (Status < ZESTETM1_WARNING_BASE && Status >= ZESTETM1_MAX_INFO))
00088     {
00089         return ZESTETM1_ILLEGAL_STATUS_CODE;
00090     }
00091     *Buffer = ZESTETM1_ERROR_STRING(Status);
00092     return ZESTETM1_SUCCESS;
00093 }
00094
00095
00096
00097

```


00098

7.25 C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/Main.c File Reference

```
#include <stdint.h>
#include <sys/socket.h>
#include "ZestETM1.h"
#include "Private.h"
```

Functions

- [ZESTETM1_STATUS ZestETM1Init \(void\)](#)
- [ZESTETM1_STATUS ZestETM1Close \(void\)](#)

7.25.1 Function Documentation

7.25.1.1 ZestETM1Close()

```
ZESTETM1_STATUS ZestETM1Close (
    void )
```

Definition at line 92 of file [Main.c](#).

```
00093 {
00094     #if defined(MSVVC) || defined(WINGCC)
00095         WSACleanup();
00096     #endif
00097
00098     return ZESTETM1_SUCCESS;
00099 }
```

7.25.1.2 ZestETM1Init()

```
ZESTETM1_STATUS ZestETM1Init (
    void )
```

Definition at line 58 of file [Main.c](#).

```
00059 {
00060     #if defined(MSVVC) || defined(WINGCC)
00061         WORD VersionRequested;
00062         WSADATA WSADATA;
00063         int Error;
00064
00065         VersionRequested = MAKEWORD(2, 2);
00066         Error = WSASStartup(VersionRequested, &WSADATA);
00067         if (Error!=0)
00068         {
00069             ZESTETM1_ERROR_GENERAL("ZestETM1Init", ZESTETM1_SOCKET_ERROR);
00070         }
00071
00072         // Confirm that the WinSock DLL supports 2.2.
00073         // Note that if the DLL supports versions greater
```

```

00074 // than 2.2 in addition to 2.2, it will still return
00075 // 2.2 in Version since that is the version we
00076 // requested.
00077 if (LOBYTE(WSAData.wVersion)!=2 ||
00078     HIBYTE(WSAData.wVersion)!=2)
00079 {
00080     WSACleanup( );
00081     ZESTETM1_ERROR_GENERAL("ZestETM1Init", ZESTETM1_SOCKET_ERROR);
00082 }
00083 #endif
00084
00085 return ZESTETM1_SUCCESS;
00086 }

```

7.26 Main.c

[Go to the documentation of this file.](#)

```

00001 // File:      Main.c
00002 //
00003 // Purpose:
00004 //     ZestETM1 Host Library
00005 //     Main functions
00006 //
00007 // Version: 1.00
00008 // Date: 11/10/12
00009
00010 // Copyright (C) 2012 Orange Tree Technologies Ltd. All rights reserved.
00011 // Orange Tree Technologies grants the purchaser of a ZestETM1 the right to use and
00012 // modify this source code in any form in designs that target the ZestETM1.
00013 // Orange Tree Technologies prohibits the use of this source code or any modification of
00014 // it in any form in designs that target any other hardware unless the purchaser of the
00015 // ZestETM1 has purchased the appropriate licence from Orange Tree Technologies.
00016 // Contact Orange Tree Technologies if you want to purchase such a licence.
00017
00018 //*****
00019 /**
00020 /*** Disclaimer: LIMITED WARRANTY AND DISCLAIMER. These designs are
00021 /*** provided to you "as is". Orange Tree Technologies and its licensors
00022 /*** make and you receive no warranties or conditions, express, implied,
00023 /*** statutory or otherwise, and Orange Tree Technologies specifically
00024 /*** disclaims any implied warranties of merchantability, non-infringement,
00025 /*** or fitness for a particular purpose. Orange Tree Technologies does not
00026 /*** warrant that the functions contained in these designs will meet your
00027 /*** requirements, or that the operation of these designs will be
00028 /*** uninterrupted or error free, or that defects in the Designs will be
00029 /*** corrected. Furthermore, Orange Tree Technologies does not warrant or
00030 /*** make any representations regarding use or the results of the use of the
00031 /*** designs in terms of correctness, accuracy, reliability, or otherwise.
00032 /***
00033 /*** LIMITATION OF LIABILITY. In no event will Orange Tree Technologies
00034 /*** or its licensors be liable for any loss of data, lost profits, cost or
00035 /*** procurement of substitute goods or services, or for any special,
00036 /*** incidental, consequential, or indirect damages arising from the use or
00037 /*** operation of the designs or accompanying documentation, however caused
00038 /*** and on any theory of liability. This limitation will apply even if
00039 /*** Orange Tree Technologies has been advised of the possibility of such
00040 /*** damage. This limitation shall apply notwithstanding the failure of the
00041 /*** essential purpose of any limited remedies herein.
00042 /***
00043 //*****
00044
00045 #include <stdint.h>
00046 #if defined(MSVC) || defined(WINGCC)
00047 #include <winsock2.h>
00048 #else
00049 #include <sys/socket.h>
00050 #endif
00051 #include "ZestETM1.h"
00052 #include "Private.h"
00053
00054 /*****
00055 * Main initialisation function.
00056 * Must be called before other ZestETM1 functions. *
00057 *****/
00058 ZESTETM1_STATUS ZestETM1Init(void)

```

```

00059 {
00060 #if defined(MSVVC) || defined(WINGCC)
00061     WORD VersionRequested;
00062     WSADATA WSADATA;
00063     int Error;
00064
00065     VersionRequested = MAKEWORD(2, 2);
00066     Error = WSStartup(VersionRequested, &WSADATA);
00067     if (Error!=0)
00068     {
00069         ZESTETM1_ERROR_GENERAL("ZestETM1Init", ZESTETM1_SOCKET_ERROR);
00070     }
00071
00072     // Confirm that the WinSock DLL supports 2.2.
00073     // Note that if the DLL supports versions greater
00074     // than 2.2 in addition to 2.2, it will still return
00075     // 2.2 in Version since that is the version we
00076     // requested.
00077     if (LOBYTE(WSADATA.wVersion)!=2 ||
00078         HIBYTE(WSADATA.wVersion)!=2)
00079     {
00080         WSACleanup( );
00081         ZESTETM1_ERROR_GENERAL("ZestETM1Init", ZESTETM1_SOCKET_ERROR);
00082     }
00083 #endif
00084
00085     return ZESTETM1_SUCCESS;
00086 }
00087
00088 /*****
00089 * Main clean up function.
00090 * Must be called after other ZestETM1 functions.
00091 *****/
00092 ZESTETM1_STATUS ZestETM1Close(void)
00093 {
00094     #if defined(MSVVC) || defined(WINGCC)
00095         WSACleanup();
00096     #endif
00097
00098     return ZESTETM1_SUCCESS;
00099 }

```

7.27 C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/Private.h File Reference

```

#include <stdint.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>

```

Classes

- struct [ZESTETM1_CONNECTION_STRUCT](#)

Macros

- #define [SD_BOTH](#) SHUT_RDWR
- #define [closesocket](#) close
- #define [ZESTETM1_ERROR](#)(f, x)

- `#define ZESTETM1_ERROR_GENERAL(f, x)`
- `#define ZESTETM1_ERROR_CONN(f, x)`
- `#define ZESTETM1_ERROR_STRING(x)`
- `#define ZESTETM1_CONNECTION_HANDLE_MAGIC 0xdeadbed1`
- `#define ZESTETM1_DEFAULT_TIMEOUT 10000`
- `#define ZESTETM1_RATE_40MHz (0<<4)`
- `#define ZESTETM1_RATE_20MHz (1<<4)`
- `#define ZESTETM1_RATE_10MHz (2<<4)`
- `#define ZESTETM1_USER_DEVICE_ID (1)`
- `#define ZESTETM1_REVERSE(x) (((x)&0xff)<<24) | (((x)&0xff00)<<8) | (((x)&0xff0000)>>8) | (((x)&0xff000000)>>24))`

Typedefs

- `typedef int SOCKET`

Functions

- `ZESTETM1_STATUS ZestETM1_OpenConnection (ZESTETM1_CARD_INFO *CardInfo, ZESTETM1_CONNECTION_TYPE Type, uint16_t Port, uint16_t LocalPort, ZESTETM1_CONNECTION *Connection)`
- `ZESTETM1_STATUS ZestETM1_CloseConnection (ZESTETM1_CONNECTION Connection)`
- `ZESTETM1_STATUS ZestETM1_SendCommand (ZESTETM1_CARD_INFO *CardInfo, ZESTETM1_CONNECTION Connection, void *WriteBuffer, uint32_t WriteLen, void *ReadBuffer, uint32_t ReadLen, int WaitForAck)`
- `ZESTETM1_STATUS ZestETM1_SPIReadWrite (ZESTETM1_CARD_INFO *CardInfo, ZESTETM1_CONNECTION Connection, int Device, int WordLen, uint32_t *WriteData, uint32_t *ReadData, uint32_t Length, int ReleaseCS, int WaitForAck)`
- `ZESTETM1_STATUS ZestETM1_WriteFlash (ZESTETM1_CARD_INFO *CardInfo, uint32_t Address, void *Buffer, uint32_t Length)`
- `ZESTETM1_STATUS ZestETM1_EraseFlashSector (ZESTETM1_CARD_INFO *CardInfo, ZESTETM1_CONNECTION Connection, uint32_t Address)`

Variables

- `ZESTETM1_ERROR_FUNC ZestETM1_ErrorHandler`
- `char * ZestETM1_ErrorStrings []`

7.27.1 Class Documentation

7.27.1.1 struct ZESTETM1_CONNECTION_STRUCT

Definition at line 93 of file [Private.h](#).

Class Members

ZESTETM1_CARD_INFO *	CardInfo	
uint16_t	LocalPort	
uint32_t	Magic	
uint16_t	Port	
SOCKET	Socket	
struct sockaddr_in	Target	
ZESTETM1_CONNECTION_TYPE	Type	

7.27.2 Macro Definition Documentation

7.27.2.1 closesocket

```
#define closesocket close
```

Definition at line 58 of file [Private.h](#).

7.27.2.2 SD_BOTH

```
#define SD_BOTH SHUT_RDWR
```

Definition at line 57 of file [Private.h](#).

7.27.2.3 ZESTETM1_CONNECTION_HANDLE_MAGIC

```
#define ZESTETM1_CONNECTION_HANDLE_MAGIC 0xdeadbed1
```

Definition at line 92 of file [Private.h](#).

7.27.2.4 ZESTETM1_DEFAULT_TIMEOUT

```
#define ZESTETM1_DEFAULT_TIMEOUT 10000
```

Definition at line 107 of file [Private.h](#).

7.27.2.5 ZESTETM1_ERROR

```
#define ZESTETM1_ERROR(  
    f,  
    x )
```

Value:

```
{ \
    if (ZestETM1_ErrorHandler!=NULL) \
        ZestETM1_ErrorHandler(f, CardInfo, x, ZESTETM1_ERROR_STRING(x)); \
    return (x); \
}
```

Definition at line 65 of file [Private.h](#).

```
00065 #define ZESTETM1_ERROR(f, x) \
00066     { \
00067         if (ZestETM1_ErrorHandler!=NULL) \
00068             ZestETM1_ErrorHandler(f, CardInfo, x, ZESTETM1_ERROR_STRING(x)); \
00069         return (x); \
00070     }
```

7.27.2.6 ZESTETM1_ERROR_CONN

```
#define ZESTETM1_ERROR_CONN(
    f,
    x )
```

Value:

```
{ \
    if (ZestETM1_ErrorHandler!=NULL) \
        ZestETM1_ErrorHandler(f, (Conn!=NULL ? Conn->CardInfo : NULL), x, ZESTETM1_ERROR_STRING(x)); \
    return (x); \
}
```

Definition at line 77 of file [Private.h](#).

```
00077 #define ZESTETM1_ERROR_CONN(f, x) \
00078     { \
00079         if (ZestETM1_ErrorHandler!=NULL) \
00080             ZestETM1_ErrorHandler(f, (Conn!=NULL ? Conn->CardInfo : NULL), x, ZESTETM1_ERROR_STRING(x)); \
00081         return (x); \
00082     }
```

7.27.2.7 ZESTETM1_ERROR_GENERAL

```
#define ZESTETM1_ERROR_GENERAL(
    f,
    x )
```

Value:

```
{ \
    if (ZestETM1_ErrorHandler!=NULL) \
        ZestETM1_ErrorHandler(f, NULL, x, ZESTETM1_ERROR_STRING(x)); \
    return (x); \
}
```

Definition at line 71 of file [Private.h](#).

```
00071 #define ZESTETM1_ERROR_GENERAL(f, x) \
00072     { \
00073         if (ZestETM1_ErrorHandler!=NULL) \
00074             ZestETM1_ErrorHandler(f, NULL, x, ZESTETM1_ERROR_STRING(x)); \
00075         return (x); \
00076     }
```

7.27.2.8 ZESTETM1_ERROR_STRING

```
#define ZESTETM1_ERROR_STRING(
    x )
```

Value:

```
ZestETM1_ErrorStrings[ (x)>=ZESTETM1_ERROR_BASE ? \
    (x)-ZESTETM1_ERROR_BASE+(ZESTETM1_MAX_INFO-ZESTETM1_INFO_BASE)+(ZESTETM1_MAX_WARNING-ZESTETM1_WARNING_BASE) : \
    ((x)>=ZESTETM1_WARNING_BASE ? \
    (x)-ZESTETM1_WARNING_BASE+(ZESTETM1_MAX_INFO-ZESTETM1_INFO_BASE) : (x)-ZESTETM1_INFO_BASE) ]
```

Definition at line 83 of file [Private.h](#).

```
00083 #define ZESTETM1_ERROR_STRING(x) \
00084     ZestETM1_ErrorStrings[ (x)>=ZESTETM1_ERROR_BASE ? \
00085         (x)-ZESTETM1_ERROR_BASE+(ZESTETM1_MAX_INFO-ZESTETM1_INFO_BASE)+(ZESTETM1_MAX_WARNING-ZESTETM1_WARNING_BASE) : \
00086         ((x)>=ZESTETM1_WARNING_BASE ? \
    (x)-ZESTETM1_WARNING_BASE+(ZESTETM1_MAX_INFO-ZESTETM1_INFO_BASE) : (x)-ZESTETM1_INFO_BASE) ]
```

7.27.2.9 ZESTETM1_RATE_10MHz

```
#define ZESTETM1_RATE_10MHz (2<<4)
```

Definition at line 112 of file [Private.h](#).

7.27.2.10 ZESTETM1_RATE_20MHz

```
#define ZESTETM1_RATE_20MHz (1<<4)
```

Definition at line 111 of file [Private.h](#).

7.27.2.11 ZESTETM1_RATE_40MHz

```
#define ZESTETM1_RATE_40MHz (0<<4)
```

Definition at line 110 of file [Private.h](#).

7.27.2.12 ZESTETM1_REVERSE

```
#define ZESTETM1_REVERSE(  
    x ) (((x)&0xff)<<24) | (((x)&0xff00)<<8) | (((x)&0xff0000)>>8) | (((x)&0xff000000)>>24)
```

Definition at line 116 of file [Private.h](#).

7.27.2.13 ZESTETM1_USER_DEVICE_ID

```
#define ZESTETM1_USER_DEVICE_ID (1)
```

Definition at line 113 of file [Private.h](#).

7.27.3 Typedef Documentation

7.27.3.1 SOCKET

```
typedef int SOCKET
```

Definition at line 56 of file [Private.h](#).

7.27.4 Function Documentation

7.27.4.1 ZestETM1_CloseConnection()

```
ZESTETM1_STATUS ZestETM1_CloseConnection (
    ZESTETM1_CONNECTION Connection )
```

Definition at line 225 of file [Data.c](#).

```
00226 {
00227     ZESTETM1_CONNECTION_STRUCT *Conn = (ZESTETM1_CONNECTION_STRUCT *)Connection;
00228
00229     if (Conn==NULL)
00230     {
00231         return ZESTETM1_NULL_PARAMETER;
00232     }
00233     if (Conn->Magic!=ZESTETM1_CONNECTION_HANDLE_MAGIC)
00234     {
00235         return ZESTETM1_ILLEGAL_CONNECTION;
00236     }
00237
00238     // Cleanup
00239     closesocket (Conn->Socket);
00240     Conn->Magic = 0;
00241     free (Conn);
00242
00243     return ZESTETM1_SUCCESS;
00244 }
```

7.27.4.2 ZestETM1_EraseFlashSector()

```
ZESTETM1_STATUS ZestETM1_EraseFlashSector (
    ZESTETM1_CARD_INFO * CardInfo,
    ZESTETM1_CONNECTION Connection,
    uint32_t Address )
```

7.27.4.3 ZestETM1_OpenConnection()

```
ZESTETM1_STATUS ZestETM1_OpenConnection (
    ZESTETM1_CARD_INFO * CardInfo,
    ZESTETM1_CONNECTION_TYPE Type,
    uint16_t Port,
    uint16_t LocalPort,
    ZESTETM1_CONNECTION * Connection )
```

Definition at line 101 of file [Data.c](#).

```
00106 {
00107     ZESTETM1_CONNECTION_STRUCT *NewStruct;
00108     SOCKET Socket = -1;
00109     char AddrBuffer[32];
00110     char PortBuffer[32];
00111
00112     if (Connection==NULL || CardInfo==NULL)
00113     {
00114         return ZESTETM1_NULL_PARAMETER;
00115     }
00116
00117     // Allocate data structure
00118     NewStruct = malloc(sizeof(ZESTETM1_CONNECTION_STRUCT));
00119     if (NewStruct==NULL)
00120     {
00121         return ZESTETM1_OUT_OF_MEMORY;
00122     }
```



```

00123
00124 // Build target addresses
00125 sprintf(AddrBuffer, "%d.%d.%d.%d", CardInfo->IPAddr[0], CardInfo->IPAddr[1],
00126         CardInfo->IPAddr[2], CardInfo->IPAddr[3]);
00127 sprintf(PortBuffer, "%d", Port);
00128
00129 if (Type==ZESTETM1_TYPE_UDP)
00130 {
00131     // Open UDP connection
00132     struct sockaddr_in SourceIP;
00133     int SourceLen = (int)sizeof(struct sockaddr_in);
00134     Socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
00135     if (Socket<0)
00136         return ZESTETM1_SOCKET_ERROR;
00137     SourceIP.sin_family = AF_INET;
00138     SourceIP.sin_addr.s_addr = htonl(INADDR_ANY);
00139     SourceIP.sin_port = htons(LocalPort);
00140     bind(Socket, (const struct sockaddr *)&SourceIP, SourceLen);
00141
00142     NewStruct->Target.sin_family = AF_INET;
00143     NewStruct->Target.sin_addr.s_addr = inet_addr(AddrBuffer);
00144     NewStruct->Target.sin_port = htons(atoi(PortBuffer));
00145 }
00146 else if (Type==ZESTETM1_TYPE_TCP)
00147 {
00148     // Open TCP connection
00149     struct addrinfo *AddrResult = NULL,
00150                 *Ptr = NULL,
00151                 Hints;
00152
00153     int Result;
00154     struct sockaddr_in SourceIP;
00155     int SourceLen = (int)sizeof(struct sockaddr_in);
00156
00157     memset(&Hints, 0, sizeof(Hints));
00158     Hints.ai_family = AF_UNSPEC;
00159     Hints.ai_socktype = SOCK_STREAM;
00160     Hints.ai_protocol = IPPROTO_TCP;
00161
00162     // Resolve the server address and port
00163     Result = getaddrinfo(AddrBuffer, PortBuffer, &Hints, &AddrResult);
00164     if (Result!=0)
00165     {
00166         free(NewStruct);
00167         return ZESTETM1_SOCKET_ERROR;
00168     }
00169
00170     // Attempt to connect to an address until one succeeds
00171     for (Ptr=AddrResult; Ptr!=NULL; Ptr=Ptr->ai_next)
00172     {
00173         // Create a SOCKET for connecting to server
00174         Socket = socket(Ptr->ai_family, Ptr->ai_socktype,
00175                        Ptr->ai_protocol);
00176         if (Socket<0)
00177         {
00178             freeaddrinfo(AddrResult);
00179             free(NewStruct);
00180             return ZESTETM1_SOCKET_ERROR;
00181         }
00182
00183         // Connect to ZestETM1
00184         Result = connect(Socket, Ptr->ai_addr, (int)Ptr->ai_addrlen);
00185         if (Result<0)
00186         {
00187             closesocket(Socket);
00188             Socket = -1;
00189             continue;
00190         }
00191         break;
00192     }
00193
00194     SourceIP.sin_family = AF_INET;
00195     SourceIP.sin_addr.s_addr = htonl(INADDR_ANY);
00196     SourceIP.sin_port = 0;
00197     bind(Socket, (const struct sockaddr *)&SourceIP, SourceLen);
00198     freeaddrinfo(AddrResult);
00199 }
00200 else
00201 {
00202     free(NewStruct);
00203     return ZESTETM1_INVALID_CONNECTION_TYPE;
00204 }

```

```

00204
00205     if (Socket== -1)
00206     {
00207         free(NewStruct);
00208         return ZESTETM1_SOCKET_ERROR;
00209     }
00210
00211     NewStruct->Magic = ZESTETM1_CONNECTION_HANDLE_MAGIC;
00212     NewStruct->Type = Type;
00213     NewStruct->Port = Port;
00214     NewStruct->LocalPort = LocalPort;
00215     NewStruct->Socket = Socket;
00216     NewStruct->CardInfo = CardInfo;
00217     *Connection = NewStruct;
00218
00219     return ZESTETM1_SUCCESS;
00220 }

```

7.27.4.4 ZestETM1_SendCommand()

```

ZESTETM1_STATUS ZestETM1_SendCommand (
    ZESTETM1_CARD_INFO * CardInfo,
    ZESTETM1_CONNECTION Connection,
    void * WriteBuffer,
    uint32_t WriteLen,
    void * ReadBuffer,
    uint32_t ReadLen,
    int WaitForAck )

```

Definition at line 453 of file [Data.c](#).

```

00458 {
00459     ZESTETM1_STATUS Result;
00460     unsigned long Written;
00461     unsigned long Received;
00462
00463     // Send/receive data
00464     Result = ZestETM1_WriteData(Connection, WriteBuffer, WriteLen, &Written,
00465                                CardInfo->Timeout);
00466     if (Result!=ZESTETM1_SUCCESS)
00467     {
00468         return Result;
00469     }
00470     if (Written!=WriteLen)
00471     {
00472         return ZESTETM1_INTERNAL_ERROR;
00473     }
00474     *((uint8_t *)ReadBuffer) = 0;
00475     if (WaitForAck==1)
00476     {
00477         Result = ZestETM1_ReadData(Connection, ReadBuffer, ReadLen, &Received,
00478                                   CardInfo->Timeout);
00479         if (Result!=ZESTETM1_SUCCESS)
00480         {
00481             return Result;
00482         }
00483         if (Received!=ReadLen)
00484         {
00485             return ZESTETM1_INTERNAL_ERROR;
00486         }
00487     }
00488
00489     return ZESTETM1_SUCCESS;
00490 }

```

7.27.4.5 ZestETM1_SPIReadWrite()

```

ZESTETM1_STATUS ZestETM1_SPIReadWrite (
    ZESTETM1_CARD_INFO * CardInfo,

```

```

ZESTETM1_CONNECTION Connection,
int Device,
int WordLen,
uint32_t * WriteData,
uint32_t * ReadData,
uint32_t Length,
int ReleaseCS,
int WaitForAck )

```

Definition at line 495 of file [Data.c](#).

```

00501 {
00502     uint8_t Buffer[65536];
00503     uint32_t *BufPtr;
00504     uint32_t i;
00505     ZESTETM1_STATUS Result;
00506
00507     // Build command
00508     Buffer[0] = ZESTETM1_COMMAND_SPI; // Command byte
00509     Buffer[1] = Device;               // SPI device
00510     Buffer[2] = WordLen;              // Word length
00511     Buffer[3] = ReleaseCS;            // Release CS on completion
00512     BufPtr = (uint32_t *) (Buffer+4);
00513
00514     BufPtr[0] = WriteData==NULL ? 0 : ZESTETM1_REVERSE(Length); // Num words
00515     BufPtr[1] = ReadData==NULL ? 0 : ZESTETM1_REVERSE(Length);
00516     if (WriteData!=NULL)
00517     {
00518         for (i=0; i<Length; i++)
00519         {
00520             BufPtr[2+i] = ZESTETM1_REVERSE(WriteData[i]);
00521         }
00522     }
00523     else
00524         memset(BufPtr+2, 0, Length*4);
00525
00526     // Send command
00527     Result = ZestETM1_SendCommand(CardInfo, Connection, Buffer,
00528                                   WriteData==NULL ? 12 : 12+Length*4,
00529                                   Buffer, ReadData==NULL ? 4 : 4+Length*4,
00530                                   WaitForAck);
00531     if (WaitForAck==0)
00532         return Result;
00533
00534     // Extract results
00535     if (Buffer[0]!=ZESTETM1_COMMAND_SPI || Buffer[1]!=0)
00536     {
00537         return ZESTETM1_INTERNAL_ERROR;
00538     }
00539
00540     BufPtr = (uint32_t *) (Buffer+4);
00541     if (ReadData!=NULL)
00542     {
00543         for (i=0; i<Length; i++)
00544         {
00545             ReadData[i] = ZESTETM1_REVERSE(BufPtr[i]);
00546         }
00547     }
00548
00549     return ZESTETM1_SUCCESS;
00550 }

```

7.27.4.6 ZestETM1_WriteFlash()

```

ZESTETM1_STATUS ZestETM1_WriteFlash (
    ZESTETM1_CARD_INFO * CardInfo,
    uint32_t Address,
    void * Buffer,
    uint32_t Length )

```

7.27.5 Variable Documentation

7.27.5.1 ZestETM1_ErrorHandler

[ZESTETM1_ERROR_FUNC](#) ZestETM1_ErrorHandler [extern]

Definition at line 67 of file [Error.c](#).

7.27.5.2 ZestETM1_ErrorStrings

char* ZestETM1_ErrorStrings[] [extern]

Definition at line 52 of file [Error.c](#).

```
00053 {
00054     "Success (no error)",
00055     "Error communicating with socket",
00056     "An unspecified internal error occurred",
00057     "Status code is out of range",
00058     "NULL was used illegally as one of the parameter values",
00059     "Not enough memory to complete the requested operation",
00060     "The requested connection type is invalid",
00061     "The requested connection is invalid",
00062     "The connection was closed unexpectedly",
00063     "Operation timed out",
00064     "One of the parameters has an illegal value",
00065
00066 };
```

7.28 Private.h

[Go to the documentation of this file.](#)

```
00001 // File:      Private.h
00002 //
00003 // Purpose:
00004 //     ZestETM1 Host Library
00005 //     Internal header file
00006 //
00007 // Version: 1.00
00008 // Date: 11/10/12
00009
00010 // Copyright (C) 2012 Orange Tree Technologies Ltd. All rights reserved.
00011 // Orange Tree Technologies grants the purchaser of a ZestETM1 the right to use and
00012 // modify this source code in any form in designs that target the ZestETM1.
00013 // Orange Tree Technologies prohibits the use of this source code or any modification of
00014 // it in any form in designs that target any other hardware unless the purchaser of the
00015 // ZestETM1 has purchased the appropriate licence from Orange Tree Technologies.
00016 // Contact Orange Tree Technologies if you want to purchase such a licence.
00017
00018 //*****
00019 /**
00020 /*** Disclaimer: LIMITED WARRANTY AND DISCLAIMER. These designs are
00021 /*** provided to you "as is". Orange Tree Technologies and its licensors
00022 /*** make and you receive no warranties or conditions, express, implied,
00023 /*** statutory or otherwise, and Orange Tree Technologies specifically
00024 /*** disclaims any implied warranties of merchantability, non-infringement,
00025 /*** or fitness for a particular purpose. Orange Tree Technologies does not
00026 /*** warrant that the functions contained in these designs will meet your
00027 /*** requirements, or that the operation of these designs will be
00028 /*** uninterrupted or error free, or that defects in the Designs will be
00029 /*** corrected. Furthermore, Orange Tree Technologies does not warrant or
00030 /*** make any representations regarding use or the results of the use of the
00031 /*** designs in terms of correctness, accuracy, reliability, or otherwise.
00032 /***
00033 /*** LIMITATION OF LIABILITY. In no event will Orange Tree Technologies
00034 /*** or its licensors be liable for any loss of data, lost profits, cost or
```

```

00035 /**      procurement of substitute goods or services, or for any special,
00036 /**      incidental, consequential, or indirect damages arising from the use or
00037 /**      operation of the designs or accompanying documentation, however caused
00038 /**      and on any theory of liability. This limitation will apply even if
00039 /**      Orange Tree Technologies has been advised of the possibility of such
00040 /**      damage. This limitation shall apply notwithstanding the failure of the
00041 /**      essential purpose of any limited remedies herein.
00042 /**
00043 /*******
00044
00045 #include <stdint.h>
00046
00047 #if defined(MSVC) || defined(WINGCC)
00048 #include "Winsock2.h"
00049 #include "Ws2tcpip.h"
00050 #else
00051 #include <unistd.h>
00052 #include <sys/socket.h>
00053 #include <netinet/in.h>
00054 #include <netdb.h>
00055 #include <arpa/inet.h>
00056 typedef int SOCKET;
00057 #define SD_BOTH SHUT_RDWR
00058 #define closesocket close
00059 #endif
00060
00061 /*****
00062 * Error macro *
00063 *****/
00064 extern ZESTETM1_ERROR_FUNC ZestETM1_ErrorHandler;
00065 #define ZESTETM1_ERROR(f, x) \
00066 { \
00067     if (ZestETM1_ErrorHandler!=NULL) \
00068         ZestETM1_ErrorHandler(f, CardInfo, x, ZESTETM1_ERROR_STRING(x)); \
00069     return (x); \
00070 }
00071 #define ZESTETM1_ERROR_GENERAL(f, x) \
00072 { \
00073     if (ZestETM1_ErrorHandler!=NULL) \
00074         ZestETM1_ErrorHandler(f, NULL, x, ZESTETM1_ERROR_STRING(x)); \
00075     return (x); \
00076 }
00077 #define ZESTETM1_ERROR_CONN(f, x) \
00078 { \
00079     if (ZestETM1_ErrorHandler!=NULL) \
00080         ZestETM1_ErrorHandler(f, (Conn!=NULL ? Conn->CardInfo : NULL), x, ZESTETM1_ERROR_STRING(x)); \
00081     return (x); \
00082 }
00083 #define ZESTETM1_ERROR_STRING(x) \
00084     ZestETM1_ErrorStrings[(x)>=ZESTETM1_ERROR_BASE ? \
00085         (x)-ZESTETM1_ERROR_BASE+(ZESTETM1_MAX_INFO-ZESTETM1_INFO_BASE)+(ZESTETM1_MAX_WARNING-ZESTETM1_WARNING_BASE) : \
00086         ((x)>=ZESTETM1_WARNING_BASE ? \
00087             (x)-ZESTETM1_WARNING_BASE+(ZESTETM1_MAX_INFO-ZESTETM1_INFO_BASE) : (x)-ZESTETM1_INFO_BASE)]
00088 extern char *ZestETM1_ErrorStrings[];
00089
00089 /*****
00090 * Network connection descriptor structure *
00091 *****/
00092 #define ZESTETM1_CONNECTION_HANDLE_MAGIC 0xdeadbed1
00093 typedef struct
00094 {
00095     uint32_t Magic;
00096     ZESTETM1_CARD_INFO *CardInfo;
00097     ZESTETM1_CONNECTION_TYPE Type;
00098     struct sockaddr_in Target;
00099     uint16_t Port;
00100     uint16_t LocalPort;
00101     SOCKET Socket;
00102 } ZESTETM1_CONNECTION_STRUCT;
00103
00104 /*****
00105 * Constants *
00106 *****/
00107 #define ZESTETM1_DEFAULT_TIMEOUT 10000
00108
00109 // SPI Device ID and clock
00110 #define ZESTETM1_RATE_40MHz (0<<4)
00111 #define ZESTETM1_RATE_20MHz (1<<4)
00112 #define ZESTETM1_RATE_10MHz (2<<4)

```

```

00113 #define ZESTETM1_USER_DEVICE_ID      (1)
00114
00115 // Reverse bytes in 32 bit word
00116 #define ZESTETM1_REVERSE(x) (((x)&0xff)<<24) | (((x)&0xff00)<<8) | (((x)&0xff0000)>>8) |
00117     (((x)&0xff000000)>>24))
00118
00118 /*****
00119  * Local functions *
00120 *****/
00121 ZESTETM1_STATUS ZestETM1_OpenConnection(ZESTETM1_CARD_INFO *CardInfo,
00122     ZESTETM1_CONNECTION_TYPE Type,
00123     uint16_t Port,
00124     uint16_t LocalPort,
00125     ZESTETM1_CONNECTION *Connection);
00126 ZESTETM1_STATUS ZestETM1_CloseConnection(ZESTETM1_CONNECTION Connection);
00127 ZESTETM1_STATUS ZestETM1_SendCommand(ZESTETM1_CARD_INFO *CardInfo,
00128     ZESTETM1_CONNECTION Connection,
00129     void *WriteBuffer, uint32_t WriteLen,
00130     void *ReadBuffer, uint32_t ReadLen,
00131     int WaitForAck);
00132 ZESTETM1_STATUS ZestETM1_SPIReadWrite(ZESTETM1_CARD_INFO *CardInfo,
00133     ZESTETM1_CONNECTION Connection,
00134     int Device,
00135     int WordLen, uint32_t *WriteData,
00136     uint32_t *ReadData, uint32_t Length,
00137     int ReleaseCS, int WaitForAck);
00138 ZESTETM1_STATUS ZestETM1_WriteFlash(ZESTETM1_CARD_INFO *CardInfo,
00139     uint32_t Address,
00140     void *Buffer,
00141     uint32_t Length);
00142 ZESTETM1_STATUS ZestETM1_EraseFlashSector(ZESTETM1_CARD_INFO *CardInfo,
00143     ZESTETM1_CONNECTION Connection,
00144     uint32_t Address);
00145
00146

```

7.29 C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/↵ UPnP.c File Reference

```

#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <memory.h>
#include "ZestETM1.h"
#include "Private.h"
#include <sys/types.h>
#include <sys/ioctl.h>
#include <arpa/inet.h>
#include <ifaddrs.h>

```

Classes

- struct [ZESTETM1_GET_SETTINGS_CMD](#)
- struct [ZESTETM1_GET_SETTINGS_RESPONSE](#)

Macros

- #define [_CRT_SECURE_NO_WARNINGS](#)
- #define [_strnicmp](#) strncasecmp
- #define [ZESTETM1_UPNP_ADDR](#) "239.255.255.250"
- #define [ZESTETM1_UPNP_PORT](#) 1900
- #define [ZESTETM1_COMMAND_GET_SETTINGS](#) 0xf0

Functions

- static int [ZestETM1_HTTPGet](#) (char *IPAddr, char *Port, char *FileName, void *Buffer, int BufferLength, int Wait)
- static [ZESTETM1_STATUS ZestETM1_ReadSettings](#) ([ZESTETM1_CARD_INFO](#) *CardInfo)
- static void [ZestETM1_GetCardInfo](#) (char *Location, uint32_t *NumCards, [ZESTETM1_CARD_INFO](#) **CardInfo, int Wait)
- static int [ZestETM1_JoinGroup](#) ([SOCKET](#) sd, uint32_t grpaddr, uint32_t iaddr)
- static int [ZestETM1_LeaveGroup](#) ([SOCKET](#) sd, uint32_t grpaddr, uint32_t iaddr)
- static [ZESTETM1_STATUS ZestETM1_GetAllAdapters](#) (uint32_t *NumAdapters, struct sockaddr_in **Adapters)
- [ZESTETM1_STATUS ZestETM1CountCards](#) (unsigned long *NumCards, [ZESTETM1_CARD_INFO](#) **CardInfo, unsigned long Wait)
- [ZESTETM1_STATUS ZestETM1FreeCards](#) ([ZESTETM1_CARD_INFO](#) *CardInfo)
- [ZESTETM1_STATUS ZestETM1GetCardInfo](#) ([ZESTETM1_CARD_INFO](#) *CardInfo)

Variables

- static char * [ZestETM1_SearchReq](#)

7.29.1 Class Documentation

7.29.1.1 struct ZESTETM1_GET_SETTINGS_CMD

Definition at line 82 of file [UPnP.c](#).

Class Members

uint8_t	Command	
uint8_t	Dummy[3]	

7.29.1.2 struct ZESTETM1_GET_SETTINGS_RESPONSE

Definition at line 87 of file [UPnP.c](#).

Class Members

uint8_t	Command	
uint16_t	ControlPort	
uint8_t	Dummy1[2]	
uint8_t	Dummy2[2]	
uint32_t	Gateway	
uint16_t	HardwareVersion	
uint16_t	HTTPPort	
uint32_t	IPAddr	
uint8_t	MACAddr[6]	
uint32_t	SerialNumber	

Class Members

uint16_t	SoftwareVersion	
uint8_t	Status	
uint32_t	SubNet	

7.29.2 Macro Definition Documentation

7.29.2.1 _CRT_SECURE_NO_WARNINGS

```
#define _CRT_SECURE_NO_WARNINGS
```

Definition at line 45 of file [UPnP.c](#).

7.29.2.2 _strnicmp

```
#define _strnicmp strncasecmp
```

Definition at line 61 of file [UPnP.c](#).

7.29.2.3 ZESTETM1_COMMAND_GET_SETTINGS

```
#define ZESTETM1_COMMAND_GET_SETTINGS 0xf0
```

Definition at line 103 of file [UPnP.c](#).

7.29.2.4 ZESTETM1_UPNP_ADDR

```
#define ZESTETM1_UPNP_ADDR "239.255.255.250"
```

Definition at line 78 of file [UPnP.c](#).

7.29.2.5 ZESTETM1_UPNP_PORT

```
#define ZESTETM1_UPNP_PORT 1900
```

Definition at line 79 of file [UPnP.c](#).

7.29.3 Function Documentation

7.29.3.1 ZestETM1_GetAllAdapters()

```
static ZESTETM1_STATUS ZestETM1_GetAllAdapters (
    uint32_t * NumAdapters,
    struct sockaddr_in ** Adapters ) [static]
```

Definition at line 431 of file UPnP.c.

```
00432 {
00433     #if defined(MSVVC) || defined(WINGCC)
00434         SOCKET Socket;
00435         SOCKET_ADDRESS_LIST *AddressListPtr;
00436         DWORD BytesRequired;
00437         int i;
00438         int Count = 0;
00439
00440         *NumAdapters = 0;
00441         *Adapters = NULL;
00442
00443         Socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
00444         if (Socket<0)
00445             return ZESTETM1_INTERNAL_ERROR;
00446
00447         WSAIoctl(Socket, SIO_ADDRESS_LIST_QUERY, NULL, 0,
00448                 NULL, 0, (LPDWORD)&BytesRequired, NULL, NULL);
00449         AddressListPtr = (SOCKET_ADDRESS_LIST *)malloc(BytesRequired);
00450         if (AddressListPtr==NULL)
00451         {
00452             closesocket(Socket);
00453             return ZESTETM1_OUT_OF_MEMORY;
00454         }
00455         if (WSAIoctl(Socket, SIO_ADDRESS_LIST_QUERY, NULL, 0,
00456                 AddressListPtr, BytesRequired, &BytesRequired, NULL, NULL)==SOCKET_ERROR)
00457         {
00458             free(AddressListPtr);
00459             closesocket(Socket);
00460             return ZESTETM1_INTERNAL_ERROR;
00461         }
00462         for (i=0; i<AddressListPtr->iAddressCount; i++)
00463         {
00464             if (AddressListPtr->Address[i].iSockaddrLength==sizeof(struct sockaddr_in))
00465             {
00466                 Count++;
00467                 (*Adapters) = (struct sockaddr_in *)realloc(*Adapters, Count*sizeof(struct sockaddr_in));
00468                 if ((*Adapters)==NULL)
00469                 {
00470                     free(AddressListPtr);
00471                     closesocket(Socket);
00472                     return ZESTETM1_OUT_OF_MEMORY;
00473                 }
00474                 memcpy(&(*Adapters)[Count-1], AddressListPtr->Address[i].lpSockaddr, sizeof(struct
00475                 sockaddr_in));
00476             }
00477         }
00478         *NumAdapters = Count;
00479         closesocket(Socket);
00480         free(AddressListPtr);
00481
00482         return ZESTETM1_SUCCESS;
00483     #else
00484         struct ifaddrs *Interfaces;
00485         struct ifaddrs *Ptr;
00486         uint32_t Count = 0;
00487
00488         if (getifaddrs(&Interfaces)!=0)
00489             return ZESTETM1_INTERNAL_ERROR;
00490
00491         *NumAdapters = 0;
00492         *Adapters = NULL;
00493
00494         Ptr = Interfaces;
00495         while (Ptr!=NULL)
00496         {
```

```

00497         if (Ptr->ifa_addr!=NULL)
00498         {
00499             Count++;
00500             (*Adapters) = (struct sockaddr_in *)realloc(*Adapters, Count*sizeof(struct sockaddr_in));
00501             if ((*Adapters)==NULL)
00502             {
00503                 freeifaddrs(Interfaces);
00504                 return ZESTETM1_OUT_OF_MEMORY;
00505             }
00506             memcpy(&(*Adapters)[Count-1], Ptr->ifa_addr, sizeof(struct sockaddr_in));
00507         }
00508         Ptr = Ptr->ifa_next;
00509     }
00510     *NumAdapters = Count;
00511     freeifaddrs(Interfaces);
00512     return ZESTETM1_SUCCESS;
00513 #endif
00514 }
00515 }

```

7.29.3.2 ZestETM1_GetCardInfo()

```

static void ZestETM1_GetCardInfo (
    char * Location,
    uint32_t * NumCards,
    ZESTETM1_CARD_INFO ** CardInfo,
    int Wait ) [static]

```

Definition at line 279 of file UPnP.c.

```

00283 {
00284     static char *Prefix = "http://";
00285     char Buffer[65536];
00286     char *IPAddr = NULL;
00287     char *FileName = NULL;
00288     char *Port = "80";
00289     int i;
00290
00291     // Extract address and port
00292     for (i=0; Location[i]!=0 && Location[i]!='\r' && Location[i]!='\n'; i++)
00293     {
00294         if (IPAddr==NULL && Prefix[i]==0) IPAddr = Location+i;
00295         if (IPAddr==NULL && Location[i]!=Prefix[i])
00296             return;
00297         if (IPAddr!=NULL)
00298         {
00299             if (Location[i]=='/' || Location[i]==0 ||
00300                 Location[i]=='\r' || Location[i]=='\n')
00301             {
00302                 Location[i]=0;
00303                 i++;
00304                 break;
00305             }
00306             if (Location[i]==':')
00307             {
00308                 int j=i;
00309                 while (Location[j]!=0 && Location[j]!='\r' &&
00310                     Location[j]!='\n' && Location[j]!='/') j++;
00311                 Location[i]=0;
00312                 Location[j]=0;
00313                 Port = Location+i+1;
00314                 i = j+1;
00315                 break;
00316             }
00317         }
00318     }
00319     if (IPAddr==NULL || Location[i]==0 || Location[i]=='\r' || Location[i]=='\n')
00320         return;
00321
00322     // Extract XML filename
00323     FileName = Location+i;
00324     while (Location[i]!=0 && Location[i]!='\r' && Location[i]!='\n') i++;
00325     Location[i] = 0;
00326 }

```

```

00327 // Get XML file
00328 memset(Buffer, 0, sizeof(Buffer));
00329 if (ZestETM1_HTTPGet(IPAddr, Port, FileName,
00330                     Buffer, sizeof(Buffer), Wait)>0)
00331 {
00332     uint8_t IPAddr[4];
00333     uint32_t i;
00334     uint16_t Port;
00335
00336     // Parse XML for service description URL
00337     char *ControlURL = strstr(Buffer, "<controlURL>");
00338     if (ControlURL==NULL)
00339         return;
00340
00341     ControlURL += 12;
00342     for (i=0; i<4; i++)
00343     {
00344         char *EndPtr;
00345         IPAddr[i] = (uint8_t)strtoul(ControlURL, &EndPtr, 10);
00346         if ((i!=3 && *EndPtr!='.') || (i==3 && *EndPtr!=':'))
00347             break;
00348         ControlURL = EndPtr+1;
00349     }
00350     if (i!=4)
00351         return;
00352
00353     Port = atoi(ControlURL);
00354
00355     // Make sure only unique devices are added to the list!
00356     if (*CardInfo!=NULL)
00357     {
00358         for (i=0; i<*NumCards; i++)
00359         {
00360             if ((*CardInfo)[i].ControlPort==Port &&
00361                 memcmp((*CardInfo)[i].IPAddr, IPAddr, sizeof(IPAddr))==0)
00362             {
00363                 break;
00364             }
00365         }
00366     }
00367     if (*CardInfo==NULL || i==*NumCards)
00368     {
00369         uint32_t Index = (*NumCards);
00370         ZESTETM1_CARD_INFO *NewBuffer;
00371
00372         // Allocate space for new card info structure
00373         if ((*CardInfo)==NULL)
00374             NewBuffer = malloc(sizeof(ZESTETM1_CARD_INFO)*(Index+1));
00375         else
00376             NewBuffer = realloc(*CardInfo, sizeof(ZESTETM1_CARD_INFO)*(Index+1));
00377
00378         // Get new card settings
00379         NewBuffer[Index].ControlPort = Port;
00380         memcpy(NewBuffer[Index].IPAddr, IPAddr, sizeof(IPAddr));
00381         NewBuffer[Index].Timeout = Wait;
00382         if (ZestETM1_ReadSettings(&(NewBuffer[Index]))!=ZESTETM1_SUCCESS)
00383         {
00384             if (*CardInfo==NULL)
00385                 free(NewBuffer);
00386             else
00387             {
00388                 (*CardInfo) = NewBuffer;
00389                 memset(NewBuffer[Index].IPAddr, 0, sizeof(NewBuffer[Index].IPAddr));
00390                 NewBuffer[Index].ControlPort = 0;
00391             }
00392         }
00393         else
00394         {
00395             NewBuffer[Index].Timeout = ZESTETM1_DEFAULT_TIMEOUT;
00396             (*CardInfo) = NewBuffer;
00397             (*NumCards)++;
00398         }
00399     }
00400 }
00401 }

```

7.29.3.3 ZestETM1_HTTPGet()

```
static int ZestETM1_HTTPGet (
    char * IPAddr,
    char * Port,
    char * FileName,
    void * Buffer,
    int BufferLength,
    int Wait ) [static]
```

Definition at line 108 of file UPnP.c.

```
00111 {
00112     struct addrinfo *Addr = NULL,
00113                 *Ptr = NULL,
00114                 Hints;
00115     int Result;
00116     SOCKET Socket;
00117     int Offset = 0;
00118     char Req[1024];
00119     char *HdrEnd;
00120     fd_set ReadFDS;
00121     struct timeval Timeout;
00122
00123     Timeout.tv_sec = Wait/1000;
00124     Timeout.tv_usec = (Wait%1000)*1000;
00125
00126     // Attempt to connect to the address
00127     memset(&Hints, 0, sizeof(Hints));
00128     Hints.ai_family = AF_UNSPEC;
00129     Hints.ai_socktype = SOCK_STREAM;
00130     Hints.ai_protocol = IPPROTO_TCP;
00131
00132     // Resolve the server address and port
00133     Result = getaddrinfo(IPAddr, Port, &Hints, &Addr);
00134     if (Result!=0)
00135         return -1;
00136
00137     for (Ptr=Addr; Ptr!=NULL; Ptr=Ptr->ai_next)
00138     {
00139         // Create a SOCKET for connecting to server
00140         Socket = socket(Ptr->ai_family, Ptr->ai_socktype,
00141                        Ptr->ai_protocol);
00142         if (Socket<0)
00143         {
00144             freeaddrinfo(Addr);
00145             return -1;
00146         }
00147
00148         // Connect to ZestETM1
00149         Result = connect(Socket, Ptr->ai_addr, (int)Ptr->ai_addrlen);
00150         if (Result<0)
00151         {
00152             closesocket(Socket);
00153             Socket = -1;
00154             continue;
00155         }
00156         break;
00157     }
00158     freeaddrinfo(Addr);
00159
00160     // Send GET request
00161     sprintf(Req, "GET /%s HTTP/1.1\r\nHOST: %s:%s\r\nContent-length: 0\r\n\r\n", FileName, IPAddr, Port);
00162     Result = send(Socket, Req, (int)strlen(Req), 0);
00163     if (Result!=strlen(Req))
00164     {
00165         closesocket(Socket);
00166         return -1;
00167     }
00168
00169     // Get response
00170     ((char*)Buffer)[0] = 0;
00171     do
00172     {
00173         FD_ZERO(&ReadFDS);
00174         FD_SET(Socket, &ReadFDS);
```

```

00175         Result = select((int)Socket+1, &ReadFDS, NULL, NULL, &Timeout);
00176         if (Result<0)
00177         {
00178             closesocket(Socket);
00179             return -1;
00180         }
00181         if (!FD_ISSET(Socket, &ReadFDS)) break;
00182         Result = recv(Socket, (char *)Buffer+Offset, BufferLength-Offset, 0);
00183         if (Result<0)
00184         {
00185             closesocket(Socket);
00186             return -1;
00187         }
00188         Offset+=Result;
00189         if (Offset==BufferLength)
00190         {
00191             break;
00192         }
00193     } while (Result!=0);
00194
00195     // Check status response
00196     if (_strnicmp("HTTP/1.1 200 OK", Buffer, 15)!=0)
00197     {
00198         closesocket(Socket);
00199         return -1;
00200     }
00201
00202     // Remove HTTP header
00203     HdrEnd = strstr(Buffer, "\r\n\r\n");
00204     if (HdrEnd==NULL)
00205     {
00206         Offset = 0;
00207     }
00208     else
00209     {
00210         Offset -= (int)(HdrEnd+4-(char*)Buffer);
00211         memcpy(Buffer, HdrEnd+4, Offset);
00212     }
00213
00214     closesocket(Socket);
00215     return Offset;
00216 }

```

7.29.3.4 ZestETM1_JoinGroup()

```

static int ZestETM1_JoinGroup (
    SOCKET sd,
    uint32_t grpaddr,
    uint32_t iaddr ) [static]

```

Definition at line 407 of file UPnP.c.

```

00409 {
00410     struct ip_mreq imr;
00411
00412     imr.imr_multiaddr.s_addr = grpaddr;
00413     imr.imr_interface.s_addr = iaddr;
00414     return setsockopt(sd, IPPROTO_IP, IP_ADD_MEMBERSHIP,
00415         (const char *)&imr, sizeof(imr));
00416 }

```

7.29.3.5 ZestETM1_LeaveGroup()

```

static int ZestETM1_LeaveGroup (
    SOCKET sd,
    uint32_t grpaddr,
    uint32_t iaddr ) [static]

```

Definition at line 417 of file UPnP.c.

```
00419 {
00420     struct ip_mreq imr;
00421
00422     imr.imr_multiaddr.s_addr = grpaddr;
00423     imr.imr_interface.s_addr = iaddr;
00424     return setsockopt(sd, IPPROTO_IP, IP_DROP_MEMBERSHIP,
00425                     (const char *)&imr, sizeof(imr));
00426 }
```

7.29.3.6 ZestETM1_ReadSettings()

```
static ZESTETM1_STATUS ZestETM1_ReadSettings (
    ZESTETM1_CARD_INFO * CardInfo ) [static]
```

Definition at line 221 of file UPnP.c.

```
00222 {
00223     ZESTETM1_STATUS Result;
00224     ZESTETM1_CONNECTION Connection;
00225     ZESTETM1_GET_SETTINGS_CMD Cmd;
00226     ZESTETM1_GET_SETTINGS_RESPONSE Response;
00227
00228     if (CardInfo==NULL)
00229     {
00230         return ZESTETM1_NULL_PARAMETER;
00231     }
00232
00233     // Open control connection
00234     Result = ZestETM1_OpenConnection(CardInfo, ZESTETM1_TYPE_TCP,
00235                                     CardInfo->ControlPort, 0, &Connection);
00236     if (Result!=ZESTETM1_SUCCESS)
00237     {
00238         return Result;
00239     }
00240
00241     // Get the settings from the device
00242     Cmd.Command = ZESTETM1_COMMAND_GET_SETTINGS;
00243     Result = ZestETM1_SendCommand(CardInfo, Connection,
00244                                 &Cmd, sizeof(Cmd),
00245                                 &Response, sizeof(Response), 1);
00246     if (Result!=ZESTETM1_SUCCESS)
00247     {
00248         ZestETM1_CloseConnection(Connection);
00249         return Result;
00250     }
00251
00252     // Result values (including endian conversion)
00253     CardInfo->FirmwareVersion = ((Response.SoftwareVersion>>8)&0xff) | ((Response.SoftwareVersion<<0xff)&8);
00254     CardInfo->HardwareVersion = ((Response.HardwareVersion>>8)&0xff) | ((Response.HardwareVersion<<0xff)&8);
00255     CardInfo->Gateway[0] = (uint8_t)((Response.Gateway>>0)&0xff);
00256     CardInfo->Gateway[1] = (uint8_t)((Response.Gateway>>8)&0xff);
00257     CardInfo->Gateway[2] = (uint8_t)((Response.Gateway>>16)&0xff);
00258     CardInfo->Gateway[3] = (uint8_t)((Response.Gateway>>24)&0xff);
00259     CardInfo->SubNet[0] = (uint8_t)((Response.SubNet>>0)&0xff);
00260     CardInfo->SubNet[1] = (uint8_t)((Response.SubNet>>8)&0xff);
00261     CardInfo->SubNet[2] = (uint8_t)((Response.SubNet>>16)&0xff);
00262     CardInfo->SubNet[3] = (uint8_t)((Response.SubNet>>24)&0xff);
00263     memcpy(CardInfo->MACAddr, Response.MACAddr, 6);
00264     CardInfo->ControlPort = ((Response.ControlPort>>8)&0xff) | ((Response.ControlPort<<0xff)&8);
00265     CardInfo->HTTPPort = ((Response.HTTPPort>>8)&0xff) | ((Response.HTTPPort<<0xff)&8);
00266     CardInfo->SerialNumber = ZESTETM1_REVERSE(Response.SerialNumber);
00267
00268     Result = ZestETM1_CloseConnection(Connection);
00269     if (Result!=ZESTETM1_SUCCESS)
00270     {
00271         return Result;
00272     }
00273     return ZESTETM1_SUCCESS;
00273 }
```

7.29.3.7 ZestETM1CountCards()

```
ZESTETM1_STATUS ZestETM1CountCards (
    unsigned long * NumCards,
```

```

    ZESTETM1_CARD_INFO ** CardInfo,
    unsigned long Wait )

```

Definition at line 521 of file UPnP.c.

```

00523 {
00524     SOCKET Socket;
00525     struct sockaddr_in DestIP;
00526     struct sockaddr_in SourceIP;
00527     int SourceIPLength;
00528     int Flag = 1;
00529     int Result;
00530     char Req[1024];
00531     char Response[1024];
00532     ZESTETM1_CARD_INFO *Cards = NULL;
00533     uint32_t CardCount = 0;
00534     int i;
00535     struct timeval Timeout;
00536     fd_set ReadFDS;
00537     uint32_t Interface;
00538     uint32_t NumAdapters;
00539     struct sockaddr_in *Adapters;
00540     ZESTETM1_STATUS Status;
00541
00542     *NumCards = 0;
00543     *CardInfo = NULL;
00544
00545     // Get a list of all adapters
00546     Status = ZestETM1_GetAllAdapters(&NumAdapters, &Adapters);
00547     if (Status!=ZESTETM1_SUCCESS)
00548         ZESTETM1_ERROR_GENERAL("ZestETM1CountCards", Status);
00549     if (NumAdapters==0)
00550     {
00551         *NumCards = 0;
00552         return ZESTETM1_SUCCESS;
00553     }
00554
00555     // Send queries on all interfaces
00556     for (Interface=0; Interface<NumAdapters; Interface++)
00557     {
00558         if (Adapters[Interface].sin_family!=AF_INET)
00559             continue;
00560
00561         // Open socket for search requests
00562         Socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
00563         if (Socket<0)
00564         {
00565             continue;
00566         }
00567
00568         // Set reuse port to on to allow multiple binds per host
00569         if (setsockopt(Socket, SOL_SOCKET, SO_REUSEADDR, (char *)&Flag,
00570             sizeof(Flag))<0)
00571         {
00572             closesocket(Socket);
00573             continue;
00574         }
00575
00576         // Bind to port for receiving responses
00577         SourceIP.sin_family = AF_INET;
00578         SourceIP.sin_addr.s_addr = htonl(INADDR_ANY);
00579         SourceIP.sin_port = 0;
00580         Result = bind(Socket, (const struct sockaddr *)(&Adapters[Interface]), sizeof(struct
sockaddr_in));
00581         if (Result<0)
00582         {
00583             closesocket(Socket);
00584             continue;
00585         }
00586
00587         // Join multicast group
00588         if (ZestETM1_JoinGroup(Socket, inet_addr(ZESTETM1_UPNP_ADDR),
00589             htonl(INADDR_ANY))<0)
00590         {
00591             closesocket(Socket);
00592             continue;
00593         }
00594
00595         // Send M-SEARCH request
00596         // Send more than once as UDP is unreliable
00597         DestIP.sin_family = AF_INET;

```

```

00598     DestIP.sin_addr.s_addr = inet_addr(ZESTETM1_UPNP_ADDR);
00599     DestIP.sin_port = htons(1900);
00600     sprintf(Req, ZestETM1_SearchReq, (Wait+999)/1000);
00601     for (i=0; i<3; i++)
00602     {
00603         Result = sendto(Socket, ZestETM1_SearchReq,
00604                             (int)strlen(ZestETM1_SearchReq),
00605                             0, (struct sockaddr *)&DestIP, sizeof(DestIP));
00606         if (Result!=strlen(ZestETM1_SearchReq))
00607         {
00608             ZestETM1_LeaveGroup(Socket, inet_addr(ZESTETM1_UPNP_ADDR),
00609                                 htonl(INADDR_ANY));
00610             closesocket(Socket);
00611             continue;
00612         }
00613     }
00614
00615     // Read responses
00616     Timeout.tv_sec = Wait/1000;
00617     Timeout.tv_usec = (Wait%1000)*1000;
00618     do
00619     {
00620         FD_ZERO(&ReadFDS);
00621         FD_SET(Socket, &ReadFDS);
00622         Result = select((int)Socket+1, &ReadFDS, NULL, NULL, &Timeout);
00623         if (Result<0)
00624         {
00625             break;
00626         }
00627         if (!FD_ISSET(Socket, &ReadFDS)) break;
00628
00629         SourceIPLength = sizeof(SourceIP);
00630         Result = recvfrom(Socket, Response, sizeof(Response),
00631                             0, (struct sockaddr *)&SourceIP, &SourceIPLength);
00632         if (Result<0)
00633         {
00634             // Error!
00635             break;
00636         }
00637         else if (Result==0)
00638         {
00639             // Clean shut down
00640             break;
00641         }
00642         else
00643         {
00644             // Parse results
00645             if (_strnicmp("NOTIFY", Response, 6)==0 ||
00646                 _strnicmp("HTTP/1.1 200 OK", Response, 15)==0)
00647             {
00648                 // Check its a GigExpedite and request XML description
00649                 char *Server = strstr(Response, "GigExpedite2");
00650                 char *Location = strstr(Response, "LOCATION");
00651                 if (Server!=NULL && Location!=NULL)
00652                 {
00653                     char *c;
00654                     for (c=Location+10; *c!=0 && *c!='\n' && *c!='\r'; c++);
00655                     *c = 0;
00656                     ZestETM1_GetCardInfo(Location+10, &CardCount, &Cards, Wait);
00657                 }
00658             }
00659         }
00660     } while(1);
00661
00662     // Leave multicast group
00663     ZestETM1_LeaveGroup(Socket, inet_addr(ZESTETM1_UPNP_ADDR),
00664                         htonl(INADDR_ANY));
00665
00666     // Close socket
00667     closesocket(Socket);
00668 }
00669
00670 *NumCards = CardCount;
00671 *CardInfo = Cards;
00672 free(Adapters);
00673
00674 return ZESTETM1_SUCCESS;
00675 }

```


7.29.3.8 ZestETM1FreeCards()

```
ZESTETM1_STATUS ZestETM1FreeCards (
    ZESTETM1_CARD_INFO * CardInfo )
```

Definition at line 680 of file [UPnP.c](#).

```
00681 {
00682     if (CardInfo!=NULL)
00683         free(CardInfo);
00684
00685     return ZESTETM1_SUCCESS;
00686 }
```

7.29.3.9 ZestETM1GetCardInfo()

```
ZESTETM1_STATUS ZestETM1GetCardInfo (
    ZESTETM1_CARD_INFO * CardInfo )
```

Definition at line 691 of file [UPnP.c](#).

```
00692 {
00693     ZESTETM1_STATUS Status;
00694
00695     Status = ZestETM1_ReadSettings(CardInfo);
00696     if (Status!=ZESTETM1_SUCCESS)
00697     {
00698         ZESTETM1_ERROR("ZestETM1GetCardInfo", Status);
00699     }
00700
00701     return ZESTETM1_SUCCESS;
00702 }
```

7.29.4 Variable Documentation

7.29.4.1 ZestETM1_SearchReq

```
char* ZestETM1_SearchReq [static]
```

Initial value:

```
=
"M-SEARCH * HTTP/1.1\r\n"
"ST: upnp:rootdevice\r\n"
"MX: %d\r\n"
"MAN: \\"ssdp:discover\\" \r\n"
"HOST: 239.255.255.250:1900\r\n"
```

Definition at line 69 of file [UPnP.c](#).

7.30 UPnP.c

[Go to the documentation of this file.](#)

```

00001 // File:      UPnP.c
00002 //
00003 // Purpose:
00004 //     ZestETM1 Host Library
00005 //     UPnP board discovery functions
00006 //
00007 // Version: 1.00
00008 // Date: 11/10/12
00009
00010 // Copyright (C) 2012 Orange Tree Technologies Ltd. All rights reserved.
00011 // Orange Tree Technologies grants the purchaser of a ZestETM1 the right to use and
00012 // modify this source code in any form in designs that target the ZestETM1.
00013 // Orange Tree Technologies prohibits the use of this source code or any modification of
00014 // it in any form in designs that target any other hardware unless the purchaser of the
00015 // ZestETM1 has purchased the appropriate licence from Orange Tree Technologies.
00016 // Contact Orange Tree Technologies if you want to purchase such a licence.
00017
00018 //*****
00019 /**
00020 /***  Disclaimer: LIMITED WARRANTY AND DISCLAIMER. These designs are
00021 /***      provided to you "as is". Orange Tree Technologies and its licensors
00022 /***      make and you receive no warranties or conditions, express, implied,
00023 /***      statutory or otherwise, and Orange Tree Technologies specifically
00024 /***      disclaims any implied warranties of merchantability, non-infringement,
00025 /***      or fitness for a particular purpose. Orange Tree Technologies does not
00026 /***      warrant that the functions contained in these designs will meet your
00027 /***      requirements, or that the operation of these designs will be
00028 /***      uninterrupted or error free, or that defects in the Designs will be
00029 /***      corrected. Furthermore, Orange Tree Technologies does not warrant or
00030 /***      make any representations regarding use or the results of the use of the
00031 /***      designs in terms of correctness, accuracy, reliability, or otherwise.
00032 /***
00033 /***      LIMITATION OF LIABILITY. In no event will Orange Tree Technologies
00034 /***      or its licensors be liable for any loss of data, lost profits, cost or
00035 /***      procurement of substitute goods or services, or for any special,
00036 /***      incidental, consequential, or indirect damages arising from the use or
00037 /***      operation of the designs or accompanying documentation, however caused
00038 /***      and on any theory of liability. This limitation will apply even if
00039 /***      Orange Tree Technologies has been advised of the possibility of such
00040 /***      damage. This limitation shall apply notwithstanding the failure of the
00041 /***      essential purpose of any limited remedies herein.
00042 /***
00043 //*****
00044
00045 #define _CRT_SECURE_NO_WARNINGS
00046 #ifdef WINGCC
00047 #define __USE_W32_SOCKETS
00048 #endif
00049
00050 #include <stdint.h>
00051 #include <stdlib.h>
00052 #include <stdio.h>
00053 #include <memory.h>
00054 #include "ZestETM1.h"
00055 #include "Private.h"
00056
00057 #ifdef WINGCC
00058 #define _strnicmp strncasecmp
00059 #endif
00060 #if !defined(MSVC) && !defined(WINGCC)
00061 #define _strnicmp strncasecmp
00062 #include <sys/types.h>
00063 #include <sys/ioctl.h>
00064 #include <arpa/inet.h>
00065 #include <ifaddrs.h>
00066 #endif
00067
00068 // UPnP search string
00069 static char *ZestETM1_SearchReq =
00070 "M-SEARCH * HTTP/1.1\r\n"
00071 "ST: upnp:rootdevice\r\n"
00072 "MX: %d\r\n"
00073 "MAN: \"ssdp:discover\"\r\n"
00074 "HOST: 239.255.255.250:1900\r\n"
00075 ;
00076

```

```

00077 // UPnP broadcast address and port
00078 #define ZESTETM1_UPNP_ADDR "239.255.255.250"
00079 #define ZESTETM1_UPNP_PORT 1900
00080
00081 // Get settings command structure
00082 typedef struct
00083 {
00084     uint8_t Command;
00085     uint8_t Dummy[3];
00086 } ZESTETM1_GET_SETTINGS_CMD;
00087 typedef struct
00088 {
00089     uint8_t Command;
00090     uint8_t Status;
00091     uint8_t Dummy1[2];
00092     uint16_t SoftwareVersion;
00093     uint16_t HardwareVersion;
00094     uint32_t SerialNumber;
00095     uint32_t IPAddr;
00096     uint32_t Gateway;
00097     uint32_t SubNet;
00098     uint16_t HTTPPort;
00099     uint16_t ControlPort;
00100     uint8_t MACAddr[6];
00101     uint8_t Dummy2[2];
00102 } ZESTETM1_GET_SETTINGS_RESPONSE;
00103 #define ZESTETM1_COMMAND_GET_SETTINGS 0xf0
00104
00105 /*****
00106 * Issue HTTP GET command *
00107 *****/
00108 static int ZestETM1_HTTPGet(char *IPAddr, char *Port,
00109                             char *FileName, void *Buffer,
00110                             int BufferLength, int Wait)
00111 {
00112     struct addrinfo *Addr = NULL,
00113                 *Ptr = NULL,
00114                 Hints;
00115     int Result;
00116     SOCKET Socket;
00117     int Offset = 0;
00118     char Req[1024];
00119     char *HdrEnd;
00120     fd_set ReadFDS;
00121     struct timeval Timeout;
00122
00123     Timeout.tv_sec = Wait/1000;
00124     Timeout.tv_usec = (Wait%1000)*1000;
00125
00126     // Attempt to connect to the address
00127     memset(&Hints, 0, sizeof(Hints));
00128     Hints.ai_family = AF_UNSPEC;
00129     Hints.ai_socktype = SOCK_STREAM;
00130     Hints.ai_protocol = IPPROTO_TCP;
00131
00132     // Resolve the server address and port
00133     Result = getaddrinfo(IPAddr, Port, &Hints, &Addr);
00134     if (Result!=0)
00135         return -1;
00136
00137     for (Ptr=Addr; Ptr!=NULL; Ptr=Ptr->ai_next)
00138     {
00139         // Create a SOCKET for connecting to server
00140         Socket = socket(Ptr->ai_family, Ptr->ai_socktype,
00141                        Ptr->ai_protocol);
00142         if (Socket<0)
00143         {
00144             freeaddrinfo(Addr);
00145             return -1;
00146         }
00147
00148         // Connect to ZestETM1
00149         Result = connect(Socket, Ptr->ai_addr, (int)Ptr->ai_addrlen);
00150         if (Result<0)
00151         {
00152             closesocket(Socket);
00153             Socket = -1;
00154             continue;
00155         }
00156         break;
00157     }

```

```

00158     freeaddrinfo(Addr);
00159
00160     // Send GET request
00161     sprintf(Req, "GET /%s HTTP/1.1\r\nHOST: %s:%s\r\nContent-length: 0\r\n\r\n", FileName, IPAddr, Port);
00162     Result = send(Socket, Req, (int)strlen(Req), 0);
00163     if (Result!=strlen(Req))
00164     {
00165         closesocket(Socket);
00166         return -1;
00167     }
00168
00169     // Get response
00170     ((char*)Buffer)[0] = 0;
00171     do
00172     {
00173         FD_ZERO(&ReadFDS);
00174         FD_SET(Socket, &ReadFDS);
00175         Result = select((int)Socket+1, &ReadFDS, NULL, NULL, &Timeout);
00176         if (Result<0)
00177         {
00178             closesocket(Socket);
00179             return -1;
00180         }
00181         if (!FD_ISSET(Socket, &ReadFDS)) break;
00182         Result = recv(Socket, (char *)Buffer+Offset, BufferLength-Offset, 0);
00183         if (Result<0)
00184         {
00185             closesocket(Socket);
00186             return -1;
00187         }
00188         Offset+=Result;
00189         if (Offset==BufferLength)
00190         {
00191             break;
00192         }
00193     } while (Result!=0);
00194
00195     // Check status response
00196     if (_strnicmp("HTTP/1.1 200 OK", Buffer, 15)!=0)
00197     {
00198         closesocket(Socket);
00199         return -1;
00200     }
00201
00202     // Remove HTTP header
00203     HdrEnd = strstr(Buffer, "\r\n\r\n");
00204     if (HdrEnd==NULL)
00205     {
00206         Offset = 0;
00207     }
00208     else
00209     {
00210         Offset -= (int)(HdrEnd+4-(char*)Buffer);
00211         memcpy(Buffer, HdrEnd+4, Offset);
00212     }
00213
00214     closesocket(Socket);
00215     return Offset;
00216 }
00217
00218 /*****
00219 * Read settings from ETM1 flash *
00220 *****/
00221 static ZESTETM1_STATUS ZestETM1_ReadSettings(ZESTETM1_CARD_INFO *CardInfo)
00222 {
00223     ZESTETM1_STATUS Result;
00224     ZESTETM1_CONNECTION Connection;
00225     ZESTETM1_GET_SETTINGS_CMD Cmd;
00226     ZESTETM1_GET_SETTINGS_RESPONSE Response;
00227
00228     if (CardInfo==NULL)
00229     {
00230         return ZESTETM1_NULL_PARAMETER;
00231     }
00232
00233     // Open control connection
00234     Result = ZestETM1_OpenConnection(CardInfo, ZESTETM1_TYPE_TCP,
00235                                     CardInfo->ControlPort, 0, &Connection);
00236     if (Result!=ZESTETM1_SUCCESS)
00237     {
00238         return Result;

```

```

00239     }
00240
00241     // Get the settings from the device
00242     Cmd.Command = ZESTETM1_COMMAND_GET_SETTINGS;
00243     Result = ZestETM1_SendCommand(CardInfo, Connection,
00244                                   &Cmd, sizeof(Cmd),
00245                                   &Response, sizeof(Response), 1);
00246     if (Result!=ZESTETM1_SUCCESS)
00247     {
00248         ZestETM1_CloseConnection(Connection);
00249         return Result;
00250     }
00251
00252     // Result values (including endian conversion)
00253     CardInfo->FirmwareVersion = ((Response.SoftwareVersion»8)&0xff) | ((Response.SoftwareVersion&0xff)«8);
00254     CardInfo->HardwareVersion = ((Response.HardwareVersion»8)&0xff) | ((Response.HardwareVersion&0xff)«8);
00255     CardInfo->Gateway[0] = (uint8_t)((Response.Gateway»0)&0xff);
00256     CardInfo->Gateway[1] = (uint8_t)((Response.Gateway»8)&0xff);
00257     CardInfo->Gateway[2] = (uint8_t)((Response.Gateway»16)&0xff);
00258     CardInfo->Gateway[3] = (uint8_t)((Response.Gateway»24)&0xff);
00259     CardInfo->SubNet[0] = (uint8_t)((Response.SubNet»0)&0xff);
00260     CardInfo->SubNet[1] = (uint8_t)((Response.SubNet»8)&0xff);
00261     CardInfo->SubNet[2] = (uint8_t)((Response.SubNet»16)&0xff);
00262     CardInfo->SubNet[3] = (uint8_t)((Response.SubNet»24)&0xff);
00263     memcpy(CardInfo->MACAddr, Response.MACAddr, 6);
00264     CardInfo->ControlPort = ((Response.ControlPort»8)&0xff) | ((Response.ControlPort&0xff)«8);
00265     CardInfo->HTTPPort = ((Response.HTTPPort»8)&0xff) | ((Response.HTTPPort&0xff)«8);
00266     CardInfo->SerialNumber = ZESTETM1_REVERSE(Response.SerialNumber);
00267
00268     Result = ZestETM1_CloseConnection(Connection);
00269     if (Result!=ZESTETM1_SUCCESS)
00270         return Result;
00271
00272     return ZESTETM1_SUCCESS;
00273 }
00274
00275
00276 /*****
00277 * Read information about a card *
00278 *****/
00279 static void ZestETM1_GetCardInfo(char *Location,
00280                                  uint32_t *NumCards,
00281                                  ZESTETM1_CARD_INFO **CardInfo,
00282                                  int Wait)
00283 {
00284     static char *Prefix = "http://";
00285     char Buffer[65536];
00286     char *IPAddr = NULL;
00287     char *FileName = NULL;
00288     char *Port = "80";
00289     int i;
00290
00291     // Extract address and port
00292     for (i=0; Location[i]!=0 && Location[i]!='\r' && Location[i]!='\n'; i++)
00293     {
00294         if (IPAddr==NULL && Prefix[i]==0) IPAddr = Location+i;
00295         if (IPAddr==NULL && Location[i]!=Prefix[i])
00296             return;
00297         if (IPAddr!=NULL)
00298         {
00299             if (Location[i]=='/' || Location[i]==0 ||
00300                 Location[i]=='\r' || Location[i]=='\n')
00301             {
00302                 Location[i]=0;
00303                 i++;
00304                 break;
00305             }
00306             if (Location[i]==':')
00307             {
00308                 int j=i;
00309                 while (Location[j]!=0 && Location[j]!='\r' &&
00310                     Location[j]!='\n' && Location[j]!='/') j++;
00311                 Location[i]=0;
00312                 Location[j]=0;
00313                 Port = Location+i+1;
00314                 i = j+1;
00315                 break;
00316             }
00317         }
00318     }
00319     if (IPAddr==NULL || Location[i]==0 || Location[i]=='\r' || Location[i]=='\n')

```

```

00320         return;
00321
00322     // Extract XML filename
00323     FileName = Location+i;
00324     while (Location[i]!=0 && Location[i]!='\r' && Location[i]!='\n') i++;
00325     Location[i] = 0;
00326
00327     // Get XML file
00328     memset(Buffer, 0, sizeof(Buffer));
00329     if (ZestETM1_HTTPGet(IPAddr, Port, FileName,
00330         Buffer, sizeof(Buffer), Wait)>0)
00331     {
00332         uint8_t IPAddr[4];
00333         uint32_t i;
00334         uint16_t Port;
00335
00336         // Parse XML for service description URL
00337         char *ControlURL = strstr(Buffer, "<controlURL>");
00338         if (ControlURL==NULL)
00339             return;
00340
00341         ControlURL += 12;
00342         for (i=0; i<4; i++)
00343         {
00344             char *EndPtr;
00345             IPAddr[i] = (uint8_t)strtoul(ControlURL, &EndPtr, 10);
00346             if ((i!=3 && *EndPtr!='.') || (i==3 && *EndPtr!=':'))
00347                 break;
00348             ControlURL = EndPtr+1;
00349         }
00350         if (i!=4)
00351             return;
00352
00353         Port = atoi(ControlURL);
00354
00355         // Make sure only unique devices are added to the list!
00356         if (*CardInfo!=NULL)
00357         {
00358             for (i=0; i<*NumCards; i++)
00359             {
00360                 if ((*CardInfo)[i].ControlPort==Port &&
00361                     memcmp((*CardInfo)[i].IPAddr, IPAddr, sizeof(IPAddr))==0)
00362                 {
00363                     break;
00364                 }
00365             }
00366         }
00367         if (*CardInfo==NULL || i==*NumCards)
00368         {
00369             uint32_t Index = (*NumCards);
00370             ZESTETM1_CARD_INFO *NewBuffer;
00371
00372             // Allocate space for new card info structure
00373             if ((*CardInfo)==NULL)
00374                 NewBuffer = malloc(sizeof(ZESTETM1_CARD_INFO)*(Index+1));
00375             else
00376                 NewBuffer = realloc(*CardInfo, sizeof(ZESTETM1_CARD_INFO)*(Index+1));
00377
00378             // Get new card settings
00379             NewBuffer[Index].ControlPort = Port;
00380             memcpy(NewBuffer[Index].IPAddr, IPAddr, sizeof(IPAddr));
00381             NewBuffer[Index].Timeout = Wait;
00382             if (ZestETM1_ReadSettings(&(NewBuffer[Index]))!=ZESTETM1_SUCCESS)
00383             {
00384                 if (*CardInfo==NULL)
00385                     free(NewBuffer);
00386                 else
00387                 {
00388                     (*CardInfo) = NewBuffer;
00389                     memset(NewBuffer[Index].IPAddr, 0, sizeof(NewBuffer[Index].IPAddr));
00390                     NewBuffer[Index].ControlPort = 0;
00391                 }
00392             }
00393             else
00394             {
00395                 NewBuffer[Index].Timeout = ZESTETM1_DEFAULT_TIMEOUT;
00396                 (*CardInfo) = NewBuffer;
00397                 (*NumCards)++;
00398             }
00399         }
00400     }

```

```

00401 }
00402
00403
00404 /*****
00405 * Multicasting functions to join and leave a group *
00406 *****/
00407 static int ZestETM1_JoinGroup(SOCKET sd, uint32_t grpaddr,
00408                               uint32_t iaddr)
00409 {
00410     struct ip_mreq imr;
00411
00412     imr.imr_multiaddr.s_addr = grpaddr;
00413     imr.imr_interface.s_addr = iaddr;
00414     return setsockopt(sd, IPPROTO_IP, IP_ADD_MEMBERSHIP,
00415                      (const char *)&imr, sizeof(imr));
00416 }
00417 static int ZestETM1_LeaveGroup(SOCKET sd, uint32_t grpaddr,
00418                               uint32_t iaddr)
00419 {
00420     struct ip_mreq imr;
00421
00422     imr.imr_multiaddr.s_addr = grpaddr;
00423     imr.imr_interface.s_addr = iaddr;
00424     return setsockopt(sd, IPPROTO_IP, IP_DROP_MEMBERSHIP,
00425                      (const char *)&imr, sizeof(imr));
00426 }
00427
00428 /*****
00429 * Get an array with all local IP addresses *
00430 *****/
00431 static ZESTETM1_STATUS ZestETM1_GetAllAdapters(uint32_t *NumAdapters, struct sockaddr_in **Adapters)
00432 {
00433     #if defined(MSVC) || defined(WINGCC)
00434     SOCKET Socket;
00435     SOCKET_ADDRESS_LIST *AddressListPtr;
00436     DWORD BytesRequired;
00437     int i;
00438     int Count = 0;
00439
00440     *NumAdapters = 0;
00441     *Adapters = NULL;
00442
00443     Socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
00444     if (Socket < 0)
00445         return ZESTETM1_INTERNAL_ERROR;
00446
00447     WSAIoctl(Socket, SIO_ADDRESS_LIST_QUERY, NULL, 0,
00448              NULL, 0, (LPDWORD)&BytesRequired, NULL, NULL);
00449     AddressListPtr = (SOCKET_ADDRESS_LIST *)malloc(BytesRequired);
00450     if (AddressListPtr == NULL)
00451     {
00452         closesocket(Socket);
00453         return ZESTETM1_OUT_OF_MEMORY;
00454     }
00455     if (WSAIoctl(Socket, SIO_ADDRESS_LIST_QUERY, NULL, 0,
00456                 AddressListPtr, BytesRequired, &BytesRequired, NULL, NULL) == SOCKET_ERROR)
00457     {
00458         free(AddressListPtr);
00459         closesocket(Socket);
00460         return ZESTETM1_INTERNAL_ERROR;
00461     }
00462
00463     for (i = 0; i < AddressListPtr->iAddressCount; i++)
00464     {
00465         if (AddressListPtr->Address[i].iSockaddrLength == sizeof(struct sockaddr_in))
00466         {
00467             Count++;
00468             (*Adapters) = (struct sockaddr_in *)realloc(*Adapters, Count * sizeof(struct sockaddr_in));
00469             if ((*Adapters) == NULL)
00470             {
00471                 free(AddressListPtr);
00472                 closesocket(Socket);
00473                 return ZESTETM1_OUT_OF_MEMORY;
00474             }
00475             memcpy(&(*Adapters)[Count-1], AddressListPtr->Address[i].lpSockaddr, sizeof(struct
sockaddr_in));
00476         }
00477     }
00478     *NumAdapters = Count;
00479     closesocket(Socket);
00480     free(AddressListPtr);

```

```

00481
00482     return ZESTETM1_SUCCESS;
00483 #else
00484     struct ifaddrs *Interfaces;
00485     struct ifaddrs *Ptr;
00486     uint32_t Count = 0;
00487
00488     if (getifaddrs(&Interfaces) != 0)
00489         return ZESTETM1_INTERNAL_ERROR;
00490
00491     *NumAdapters = 0;
00492     *Adapters = NULL;
00493
00494     Ptr = Interfaces;
00495     while (Ptr != NULL)
00496     {
00497         if (Ptr->ifa_addr != NULL)
00498         {
00499             Count++;
00500             (*Adapters) = (struct sockaddr_in *)realloc(*Adapters, Count*sizeof(struct sockaddr_in));
00501             if ((*Adapters) == NULL)
00502             {
00503                 freeifaddrs(Interfaces);
00504                 return ZESTETM1_OUT_OF_MEMORY;
00505             }
00506             memcpy(&(*Adapters)[Count-1], Ptr->ifa_addr, sizeof(struct sockaddr_in));
00507         }
00508         Ptr = Ptr->ifa_next;
00509     }
00510     *NumAdapters = Count;
00511     freeifaddrs(Interfaces);
00512     return ZESTETM1_SUCCESS;
00513 #endif
00514 }
00515
00516
00517 /*****
00518 * Scan networks for ZestETM1 cards and return the number of attached devices *
00519 * and details about each one *
00520 *****/
00521 ZESTETM1_STATUS ZestETM1CountCards(unsigned long *NumCards,
00522                                     ZESTETM1_CARD_INFO **CardInfo, unsigned long Wait)
00523 {
00524     SOCKET Socket;
00525     struct sockaddr_in DestIP;
00526     struct sockaddr_in SourceIP;
00527     int SourceIPLength;
00528     int Flag = 1;
00529     int Result;
00530     char Req[1024];
00531     char Response[1024];
00532     ZESTETM1_CARD_INFO *Cards = NULL;
00533     uint32_t CardCount = 0;
00534     int i;
00535     struct timeval Timeout;
00536     fd_set ReadFDS;
00537     uint32_t Interface;
00538     uint32_t NumAdapters;
00539     struct sockaddr_in *Adapters;
00540     ZESTETM1_STATUS Status;
00541
00542     *NumCards = 0;
00543     *CardInfo = NULL;
00544
00545     // Get a list of all adapters
00546     Status = ZestETM1_GetAllAdapters(&NumAdapters, &Adapters);
00547     if (Status != ZESTETM1_SUCCESS)
00548         ZESTETM1_ERROR_GENERAL("ZestETM1CountCards", Status);
00549     if (NumAdapters == 0)
00550     {
00551         *NumCards = 0;
00552         return ZESTETM1_SUCCESS;
00553     }
00554
00555     // Send queries on all interfaces
00556     for (Interface = 0; Interface < NumAdapters; Interface++)
00557     {
00558         if (Adapters[Interface].sin_family != AF_INET)
00559             continue;
00560
00561         // Open socket for search requests

```



```

00562     Socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
00563     if (Socket<0)
00564     {
00565         continue;
00566     }
00567
00568     // Set reuse port to on to allow multiple binds per host
00569     if (setsockopt(Socket, SOL_SOCKET, SO_REUSEADDR, (char *)&Flag,
00570         sizeof(Flag))<0)
00571     {
00572         closesocket(Socket);
00573         continue;
00574     }
00575
00576     // Bind to port for receiving responses
00577     SourceIP.sin_family = AF_INET;
00578     SourceIP.sin_addr.s_addr = htonl(INADDR_ANY);
00579     SourceIP.sin_port = 0;
00580     Result = bind(Socket, (const struct sockaddr *)&Adapters[Interface], sizeof(struct
sockaddr_in));
00581     if (Result<0)
00582     {
00583         closesocket(Socket);
00584         continue;
00585     }
00586
00587     // Join multicast group
00588     if (ZestETM1_JoinGroup(Socket, inet_addr(ZESTETM1_UPNP_ADDR),
00589         htonl(INADDR_ANY))<0)
00590     {
00591         closesocket(Socket);
00592         continue;
00593     }
00594
00595     // Send M-SEARCH request
00596     // Send more than once as UDP is unreliable
00597     DestIP.sin_family = AF_INET;
00598     DestIP.sin_addr.s_addr = inet_addr(ZESTETM1_UPNP_ADDR);
00599     DestIP.sin_port = htons(1900);
00600     sprintf(Req, ZestETM1_SearchReq, (Wait+999)/1000);
00601     for (i=0; i<3; i++)
00602     {
00603         Result = sendto(Socket, ZestETM1_SearchReq,
00604             (int)strlen(ZestETM1_SearchReq),
00605             0, (struct sockaddr *)&DestIP, sizeof(DestIP));
00606         if (Result!=strlen(ZestETM1_SearchReq))
00607         {
00608             ZestETM1_LeaveGroup(Socket, inet_addr(ZESTETM1_UPNP_ADDR),
00609                 htonl(INADDR_ANY));
00610             closesocket(Socket);
00611             continue;
00612         }
00613     }
00614
00615     // Read responses
00616     Timeout.tv_sec = Wait/1000;
00617     Timeout.tv_usec = (Wait%1000)*1000;
00618     do
00619     {
00620         FD_ZERO(&ReadFDS);
00621         FD_SET(Socket, &ReadFDS);
00622         Result = select((int)Socket+1, &ReadFDS, NULL, NULL, &Timeout);
00623         if (Result<0)
00624         {
00625             break;
00626         }
00627         if (!FD_ISSET(Socket, &ReadFDS)) break;
00628
00629         SourceIPLength = sizeof(SourceIP);
00630         Result = recvfrom(Socket, Response, sizeof(Response),
00631             0, (struct sockaddr *)&SourceIP, &SourceIPLength);
00632         if (Result<0)
00633         {
00634             // Error!
00635             break;
00636         }
00637         else if (Result==0)
00638         {
00639             // Clean shut down
00640             break;
00641         }

```

```

00642         else
00643         {
00644             // Parse results
00645             if (_strnicmp("NOTIFY", Response, 6)==0 ||
00646                 _strnicmp("HTTP/1.1 200 OK", Response, 15)==0)
00647             {
00648                 // Check its a GigExpedite and request XML description
00649                 char *Server = strstr(Response, "GigExpedite2");
00650                 char *Location = strstr(Response, "LOCATION");
00651                 if (Server!=NULL && Location!=NULL)
00652                 {
00653                     char *c;
00654                     for (c=Location+10; *c!=0 && *c!='\n' && *c!='\r'; c++);
00655                     *c = 0;
00656                     ZestETM1_GetCardInfo(Location+10, &CardCount, &Cards, Wait);
00657                 }
00658             }
00659         } while(1);
00660     } while(1);
00661
00662     // Leave multicast group
00663     ZestETM1_LeaveGroup(Socket, inet_addr(ZESTETM1_UPNP_ADDR),
00664                       htonl(INADDR_ANY));
00665
00666     // Close socket
00667     closesocket(Socket);
00668 }
00669
00670 *NumCards = CardCount;
00671 *CardInfo = Cards;
00672 free(Adapters);
00673
00674 return ZESTETM1_SUCCESS;
00675 }
00676
00677 /*****
00678 * Free data structures returned by ZestETM1CountCards *
00679 *****/
00680 ZESTETM1_STATUS ZestETM1FreeCards(ZESTETM1_CARD_INFO *CardInfo)
00681 {
00682     if (CardInfo!=NULL)
00683         free(CardInfo);
00684
00685     return ZESTETM1_SUCCESS;
00686 }
00687
00688 /*****
00689 * Fill in card information fields *
00690 *****/
00691 ZESTETM1_STATUS ZestETM1GetCardInfo(ZESTETM1_CARD_INFO *CardInfo)
00692 {
00693     ZESTETM1_STATUS Status;
00694
00695     Status = ZestETM1_ReadSettings(CardInfo);
00696     if (Status!=ZESTETM1_SUCCESS)
00697     {
00698         ZESTETM1_ERROR("ZestETM1GetCardInfo", Status);
00699     }
00700
00701     return ZESTETM1_SUCCESS;
00702 }
00703

```

7.31 C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/ZestETM1.h File Reference

Classes

- struct [ZESTETM1_CARD_INFO](#)

Macros

- #define `ZESTETM1_VERSION_FALLBACK` 0x8000
- #define `ZESTETM1_INFO_BASE` 0
- #define `ZESTETM1_WARNING_BASE` 0x4000
- #define `ZESTETM1_ERROR_BASE` 0x8000

Typedefs

- typedef void * `ZESTETM1_HANDLE`
- typedef void * `ZESTETM1_CONNECTION`
- typedef void(* `ZESTETM1_ERROR_FUNC`) (const char *Function, `ZESTETM1_CARD_INFO` *CardInfo, `ZESTETM1_STATUS` Status, const char *Msg)

Enumerations

- enum `ZESTETM1_CONNECTION_TYPE` { `ZESTETM1_TYPE_TCP` , `ZESTETM1_TYPE_UDP` }
- enum `ZESTETM1_SPI_RATE` { `ZESTETM1_SPI_RATE_35` , `ZESTETM1_SPI_RATE_17_5` , `ZESTETM1_SPI_RATE_8_75` }
- enum `ZESTETM1_STATUS` {
`ZESTETM1_SUCCESS` = `ZESTETM1_INFO_BASE` , `ZESTETM1_MAX_INFO` , `ZESTETM1_MAX_WARNING`
= `ZESTETM1_WARNING_BASE` , `ZESTETM1_SOCKET_ERROR` = `ZESTETM1_ERROR_BASE` ,
`ZESTETM1_INTERNAL_ERROR` , `ZESTETM1_ILLEGAL_STATUS_CODE` , `ZESTETM1_NULL_PARAMETER` ,
`ZESTETM1_OUT_OF_MEMORY` ,
`ZESTETM1_INVALID_CONNECTION_TYPE` , `ZESTETM1_ILLEGAL_CONNECTION` , `ZESTETM1_SOCKET_CLOSED`
, `ZESTETM1_TIMEOUT` ,
`ZESTETM1_ILLEGAL_PARAMETER` , `ZESTETM1_MAX_ERROR` }

Functions

- `ZESTETM1_STATUS` `ZestETM1Init` (void)
- `ZESTETM1_STATUS` `ZestETM1Close` (void)
- `ZESTETM1_STATUS` `ZestETM1CountCards` (unsigned long *NumCards, `ZESTETM1_CARD_INFO` **CardInfo, unsigned long Wait)
- `ZESTETM1_STATUS` `ZestETM1GetCardInfo` (`ZESTETM1_CARD_INFO` *CardInfo)
- `ZESTETM1_STATUS` `ZestETM1FreeCards` (`ZESTETM1_CARD_INFO` *CardInfo)
- `ZESTETM1_STATUS` `ZestETM1RegisterErrorHandler` (`ZESTETM1_ERROR_FUNC` Function)
- `ZESTETM1_STATUS` `ZestETM1GetErrorMessage` (`ZESTETM1_STATUS` Status, char **Buffer)
- `ZESTETM1_STATUS` `ZestETM1OpenConnection` (`ZESTETM1_CARD_INFO` *CardInfo, `ZESTETM1_CONNECTION_TYPE` Type, unsigned short Port, unsigned short LocalPort, `ZESTETM1_CONNECTION` *Connection)
- `ZESTETM1_STATUS` `ZestETM1CloseConnection` (`ZESTETM1_CONNECTION` Connection)
- `ZESTETM1_STATUS` `ZestETM1WriteData` (`ZESTETM1_CONNECTION` Connection, void *Buffer, unsigned long Length, unsigned long *Written, unsigned long Timeout)
- `ZESTETM1_STATUS` `ZestETM1ReadData` (`ZESTETM1_CONNECTION` Connection, void *Buffer, unsigned long Length, unsigned long *Read, unsigned long Timeout)
- `ZESTETM1_STATUS` `ZestETM1SPIReadWrite` (`ZESTETM1_CARD_INFO` *CardInfo, `ZESTETM1_SPI_RATE` Rate, int WordLen, void *WriteData, void *ReadData, unsigned long Length, int ReleaseCS)
- `ZESTETM1_STATUS` `ZestETM1WriteRegister` (`ZESTETM1_CARD_INFO` *CardInfo, unsigned long Addr, unsigned short Data)
- `ZESTETM1_STATUS` `ZestETM1ReadRegister` (`ZESTETM1_CARD_INFO` *CardInfo, unsigned long Addr, unsigned short *Data)
- `ZESTETM1_STATUS` `ZestETM1SetInterrupt` (`ZESTETM1_CARD_INFO` *CardInfo)

7.31.1 Class Documentation

7.31.1.1 struct ZESTETM1_CARD_INFO

Definition at line 30 of file [ZestETM1.h](#).

Class Members

unsigned short	ControlPort	
unsigned long	FirmwareVersion	
unsigned char	Gateway[4]	
unsigned long	HardwareVersion	
unsigned short	HTTPPort	
unsigned char	IPAddr[4]	
unsigned char	MACAddr[6]	
unsigned long	SerialNumber	
unsigned char	SubNet[4]	
unsigned long	Timeout	

7.31.2 Macro Definition Documentation

7.31.2.1 ZESTETM1_ERROR_BASE

```
#define ZESTETM1_ERROR_BASE 0x8000
```

Definition at line 77 of file [ZestETM1.h](#).

7.31.2.2 ZESTETM1_INFO_BASE

```
#define ZESTETM1_INFO_BASE 0
```

Definition at line 75 of file [ZestETM1.h](#).

7.31.2.3 ZESTETM1_VERSION_FALLBACK

```
#define ZESTETM1_VERSION_FALLBACK 0x8000
```

Definition at line 50 of file [ZestETM1.h](#).

7.31.2.4 ZESTETM1_WARNING_BASE

```
#define ZESTETM1_WARNING_BASE 0x4000
```

Definition at line 76 of file [ZestETM1.h](#).

7.31.3 Typedef Documentation

7.31.3.1 ZESTETM1_CONNECTION

```
typedef void* ZESTETM1_CONNECTION
```

Definition at line 55 of file [ZestETM1.h](#).

7.31.3.2 ZESTETM1_ERROR_FUNC

```
typedef void(* ZESTETM1_ERROR_FUNC) (const char *Function, ZESTETM1_CARD_INFO *CardInfo, ZESTETM1_STATUS  
Status, const char *Msg)
```

Definition at line 98 of file [ZestETM1.h](#).

7.31.3.3 ZESTETM1_HANDLE

```
typedef void* ZESTETM1_HANDLE
```

Definition at line 24 of file [ZestETM1.h](#).

7.31.4 Enumeration Type Documentation

7.31.4.1 ZESTETM1_CONNECTION_TYPE

```
enum ZESTETM1_CONNECTION_TYPE
```

Enumerator

ZESTETM1_TYPE_TCP	
ZESTETM1_TYPE_UDP	

Definition at line 56 of file [ZestETM1.h](#).

```
00057 {  
00058     ZESTETM1_TYPE_TCP,  
00059     ZESTETM1_TYPE_UDP  
00060 } ZESTETM1_CONNECTION_TYPE;
```

7.31.4.2 ZESTETM1_SPI_RATE

```
enum ZESTETM1_SPI_RATE
```

Enumerator

ZESTETM1_SPI_RATE_35	
----------------------	--

Enumerator

ZESTETM1_SPI_RATE_17↔ _5	
ZESTETM1_SPI_RATE_8_75	

Definition at line 65 of file [ZestETM1.h](#).

```
00066 {
00067     ZESTETM1_SPI_RATE_35,
00068     ZESTETM1_SPI_RATE_17_5,
00069     ZESTETM1_SPI_RATE_8_75,
00070 } ZESTETM1_SPI_RATE;
```

7.31.4.3 ZESTETM1_STATUS

enum [ZESTETM1_STATUS](#)

Enumerator

ZESTETM1_SUCCESS	
ZESTETM1_MAX_INFO	
ZESTETM1_MAX_WARNING	
ZESTETM1_SOCKET_ERROR	
ZESTETM1_INTERNAL_ERROR	
ZESTETM1_ILLEGAL_STATUS_CODE	
ZESTETM1_NULL_PARAMETER	
ZESTETM1_OUT_OF_MEMORY	
ZESTETM1_INVALID_CONNECTION_TYPE	
ZESTETM1_ILLEGAL_CONNECTION	
ZESTETM1_SOCKET_CLOSED	
ZESTETM1_TIMEOUT	
ZESTETM1_ILLEGAL_PARAMETER	
ZESTETM1_MAX_ERROR	

Definition at line 78 of file [ZestETM1.h](#).

```
00079 {
00080     ZESTETM1_SUCCESS = ZESTETM1_INFO_BASE,
00081     ZESTETM1_MAX_INFO,
00082
00083     ZESTETM1_MAX_WARNING = ZESTETM1_WARNING_BASE,
00084
00085     ZESTETM1_SOCKET_ERROR = ZESTETM1_ERROR_BASE,
00086     ZESTETM1_INTERNAL_ERROR,
00087     ZESTETM1_ILLEGAL_STATUS_CODE,
00088     ZESTETM1_NULL_PARAMETER,
00089     ZESTETM1_OUT_OF_MEMORY,
00090     ZESTETM1_INVALID_CONNECTION_TYPE,
00091     ZESTETM1_ILLEGAL_CONNECTION,
00092     ZESTETM1_SOCKET_CLOSED,
00093     ZESTETM1_TIMEOUT,
00094     ZESTETM1_ILLEGAL_PARAMETER,
00095
00096     ZESTETM1_MAX_ERROR
00097 } ZESTETM1_STATUS;
```

7.31.5 Function Documentation

7.31.5.1 ZestETM1Close()

```
ZESTETM1_STATUS ZestETM1Close (
    void )
```

Definition at line 92 of file [Main.c](#).

```
00093 {
00094     #if defined(MSVC) || defined(WINGCC)
00095         WSACleanup();
00096     #endif
00097
00098     return ZESTETM1_SUCCESS;
00099 }
```

7.31.5.2 ZestETM1CloseConnection()

```
ZESTETM1_STATUS ZestETM1CloseConnection (
    ZESTETM1_CONNECTION Connection )
```

Definition at line 575 of file [Data.c](#).

```
00576 {
00577     ZESTETM1_STATUS Result;
00578     ZESTETM1_CONNECTION_STRUCT *Conn = (ZESTETM1_CONNECTION_STRUCT*)Connection;
00579
00580     Result = ZestETM1_CloseConnection(Connection);
00581     if (Result != ZESTETM1_SUCCESS)
00582     {
00583         ZESTETM1_ERROR_CONN("ZestETM1CloseConnection", Result);
00584     }
00585
00586     return ZESTETM1_SUCCESS;
00587 }
```

7.31.5.3 ZestETM1CountCards()

```
ZESTETM1_STATUS ZestETM1CountCards (
    unsigned long * NumCards,
    ZESTETM1_CARD_INFO ** CardInfo,
    unsigned long Wait )
```

Definition at line 521 of file [UPnP.c](#).

```
00523 {
00524     SOCKET Socket;
00525     struct sockaddr_in DestIP;
00526     struct sockaddr_in SourceIP;
00527     int SourceIPLength;
00528     int Flag = 1;
00529     int Result;
00530     char Req[1024];
00531     char Response[1024];
00532     ZESTETM1_CARD_INFO *Cards = NULL;
00533     uint32_t CardCount = 0;
00534     int i;
00535     struct timeval Timeout;
00536     fd_set ReadFDS;
00537     uint32_t Interface;
00538     uint32_t NumAdapters;
00539     struct sockaddr_in *Adapters;
00540     ZESTETM1_STATUS Status;
00541
00542     *NumCards = 0;
```

```

00543     *CardInfo = NULL;
00544
00545     // Get a list of all adapters
00546     Status = ZestETM1_GetAllAdapters(&NumAdapters, &Adapters);
00547     if (Status!=ZESTETM1_SUCCESS)
00548         ZESTETM1_ERROR_GENERAL("ZestETM1CountCards", Status);
00549     if (NumAdapters==0)
00550     {
00551         *NumCards = 0;
00552         return ZESTETM1_SUCCESS;
00553     }
00554
00555     // Send queries on all interfaces
00556     for (Interface=0; Interface<NumAdapters; Interface++)
00557     {
00558         if (Adapters[Interface].sin_family!=AF_INET)
00559             continue;
00560
00561         // Open socket for search requests
00562         Socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
00563         if (Socket<0)
00564         {
00565             continue;
00566         }
00567
00568         // Set reuse port to on to allow multiple binds per host
00569         if (setsockopt(Socket, SOL_SOCKET, SO_REUSEADDR, (char *)&Flag,
00570             sizeof(Flag))<0)
00571         {
00572             closesocket(Socket);
00573             continue;
00574         }
00575
00576         // Bind to port for receiving responses
00577         SourceIP.sin_family = AF_INET;
00578         SourceIP.sin_addr.s_addr = htonl(INADDR_ANY);
00579         SourceIP.sin_port = 0;
00580         Result = bind(Socket, (const struct sockaddr *)&Adapters[Interface], sizeof(struct
sockaddr_in));
00581         if (Result<0)
00582         {
00583             closesocket(Socket);
00584             continue;
00585         }
00586
00587         // Join multicast group
00588         if (ZestETM1_JoinGroup(Socket, inet_addr(ZESTETM1_UPNP_ADDR),
00589             htonl(INADDR_ANY))<0)
00590         {
00591             closesocket(Socket);
00592             continue;
00593         }
00594
00595         // Send M-SEARCH request
00596         // Send more than once as UDP is unreliable
00597         DestIP.sin_family = AF_INET;
00598         DestIP.sin_addr.s_addr = inet_addr(ZESTETM1_UPNP_ADDR);
00599         DestIP.sin_port = htons(1900);
00600         sprintf(Req, ZestETM1_SearchReq, (Wait+999)/1000);
00601         for (i=0; i<3; i++)
00602         {
00603             Result = sendto(Socket, ZestETM1_SearchReq,
00604                 (int)strlen(ZestETM1_SearchReq),
00605                 0, (struct sockaddr *)&DestIP, sizeof(DestIP));
00606             if (Result!=strlen(ZestETM1_SearchReq))
00607             {
00608                 ZestETM1_LeaveGroup(Socket, inet_addr(ZESTETM1_UPNP_ADDR),
00609                     htonl(INADDR_ANY));
00610                 closesocket(Socket);
00611                 continue;
00612             }
00613         }
00614
00615         // Read responses
00616         Timeout.tv_sec = Wait/1000;
00617         Timeout.tv_usec = (Wait%1000)*1000;
00618         do
00619         {
00620             FD_ZERO(&ReadFDS);
00621             FD_SET(Socket, &ReadFDS);
00622             Result = select((int)Socket+1, &ReadFDS, NULL, NULL, &Timeout);

```



```

00623         if (Result<0)
00624         {
00625             break;
00626         }
00627         if (!FD_ISSET(Socket, &ReadFDS)) break;
00628
00629         SourceIPLength = sizeof(SourceIP);
00630         Result = recvfrom(Socket, Response, sizeof(Response),
00631             0, (struct sockaddr *)&SourceIP, &SourceIPLength);
00632         if (Result<0)
00633         {
00634             // Error!
00635             break;
00636         }
00637         else if (Result==0)
00638         {
00639             // Clean shut down
00640             break;
00641         }
00642         else
00643         {
00644             // Parse results
00645             if (_strnicmp("NOTIFY", Response, 6)==0 ||
00646                 _strnicmp("HTTP/1.1 200 OK", Response, 15)==0)
00647             {
00648                 // Check its a GigExpedite and request XML description
00649                 char *Server = strstr(Response, "GigExpedite2");
00650                 char *Location = strstr(Response, "LOCATION");
00651                 if (Server!=NULL && Location!=NULL)
00652                 {
00653                     char *c;
00654                     for (c=Location+10; *c!=0 && *c!='\n' && *c!='\r'; c++);
00655                     *c = 0;
00656                     ZestETM1_GetCardInfo(Location+10, &CardCount, &Cards, Wait);
00657                 }
00658             }
00659         }
00660     } while(1);
00661
00662     // Leave multicast group
00663     ZestETM1_LeaveGroup(Socket, inet_addr(ZESTETM1_UPNP_ADDR),
00664         htonl(INADDR_ANY));
00665
00666     // Close socket
00667     closesocket(Socket);
00668 }
00669
00670 *NumCards = CardCount;
00671 *CardInfo = Cards;
00672 free(Adapters);
00673
00674 return ZESTETM1_SUCCESS;
00675 }

```

7.31.5.4 ZestETM1FreeCards()

```

ZESTETM1_STATUS ZestETM1FreeCards (
    ZESTETM1_CARD_INFO * CardInfo )

```

Definition at line 680 of file UPnP.c.

```

00681 {
00682     if (CardInfo!=NULL)
00683         free(CardInfo);
00684
00685     return ZESTETM1_SUCCESS;
00686 }

```

7.31.5.5 ZestETM1GetCardInfo()

```

ZESTETM1_STATUS ZestETM1GetCardInfo (
    ZESTETM1_CARD_INFO * CardInfo )

```

Definition at line 691 of file [UPnP.c](#).

```
00692 {
00693     ZESTETM1_STATUS Status;
00694
00695     Status = ZestETM1_ReadSettings(CardInfo);
00696     if (Status!=ZESTETM1_SUCCESS)
00697     {
00698         ZESTETM1_ERROR("ZestETM1GetCardInfo", Status);
00699     }
00700
00701     return ZESTETM1_SUCCESS;
00702 }
```

7.31.5.6 ZestETM1GetErrorMessage()

```
ZESTETM1_STATUS ZestETM1GetErrorMessage (
    ZESTETM1_STATUS Status,
    char ** Buffer )
```

Definition at line 84 of file [Error.c](#).

```
00086 {
00087     if (Status>ZESTETM1_MAX_ERROR ||
00088         (Status<ZESTETM1_ERROR_BASE && Status>=ZESTETM1_MAX_WARNING) ||
00089         (Status<ZESTETM1_WARNING_BASE && Status>=ZESTETM1_MAX_INFO))
00090     {
00091         return ZESTETM1_ILLEGAL_STATUS_CODE;
00092     }
00093
00094     *Buffer = ZESTETM1_ERROR_STRING(Status);
00095     return ZESTETM1_SUCCESS;
00096 }
```

7.31.5.7 ZestETM1Init()

```
ZESTETM1_STATUS ZestETM1Init (
    void )
```

Definition at line 58 of file [Main.c](#).

```
00059 {
00060     #if defined(MSVC) || defined(WINGCC)
00061         WORD VersionRequested;
00062         WSADATA WSADATA;
00063         int Error;
00064
00065         VersionRequested = MAKEWORD(2, 2);
00066         Error = WSASStartup(VersionRequested, &WSADATA);
00067         if (Error!=0)
00068         {
00069             ZESTETM1_ERROR_GENERAL("ZestETM1Init", ZESTETM1_SOCKET_ERROR);
00070         }
00071
00072         // Confirm that the WinSock DLL supports 2.2.
00073         // Note that if the DLL supports versions greater
00074         // than 2.2 in addition to 2.2, it will still return
00075         // 2.2 in Version since that is the version we
00076         // requested.
00077         if (LOBYTE(WSADATA.wVersion)!=2 ||
00078             HIBYTE(WSADATA.wVersion)!=2)
00079         {
00080             WSACleanup( );
00081             ZESTETM1_ERROR_GENERAL("ZestETM1Init", ZESTETM1_SOCKET_ERROR);
00082         }
00083     #endif
00084
00085     return ZESTETM1_SUCCESS;
00086 }
```

7.31.5.8 ZestETM1OpenConnection()

```

ZESTETM1_STATUS ZestETM1OpenConnection (
    ZESTETM1_CARD_INFO * CardInfo,
    ZESTETM1_CONNECTION_TYPE Type,
    unsigned short Port,
    unsigned short LocalPort,
    ZESTETM1_CONNECTION * Connection )

```

7.31.5.9 ZestETM1ReadData()

```

ZESTETM1_STATUS ZestETM1ReadData (
    ZESTETM1_CONNECTION Connection,
    void * Buffer,
    unsigned long Length,
    unsigned long * Read,
    unsigned long Timeout )

```

Definition at line 613 of file [Data.c](#).

```

00618 {
00619     ZESTETM1_STATUS Result;
00620     ZESTETM1_CONNECTION_STRUCT *Conn = (ZESTETM1_CONNECTION_STRUCT*)Connection;
00621
00622     Result = ZestETM1_ReadData(Connection, Buffer, Length, Read, Timeout);
00623     if (Result!=ZESTETM1_SUCCESS)
00624     {
00625         ZESTETM1_ERROR_CONN("ZestETM1ReadData", Result);
00626     }
00627
00628     return ZESTETM1_SUCCESS;
00629 }

```

7.31.5.10 ZestETM1ReadRegister()

```

ZESTETM1_STATUS ZestETM1ReadRegister (
    ZESTETM1_CARD_INFO * CardInfo,
    unsigned long Addr,
    unsigned short * Data )

```

Definition at line 728 of file [Data.c](#).

```

00729 {
00730     ZESTETM1_STATUS Result;
00731     ZESTETM1_CONNECTION Connection;
00732     ZESTETM1_READ_REG_CMD Cmd;
00733     ZESTETM1_READ_REG_RESPONSE Response;
00734
00735     if (CardInfo==NULL || Data==NULL)
00736     {
00737         ZESTETM1_ERROR("ZestETM1ReadRegister", ZESTETM1_NULL_PARAMETER);
00738     }
00739     if (Addr>127)
00740     {
00741         ZESTETM1_ERROR("ZestETM1ReadRegister", ZESTETM1_ILLEGAL_PARAMETER);
00742     }
00743
00744     Result = ZestETM1_OpenConnection(CardInfo, ZESTETM1_TYPE_TCP, CardInfo->ControlPort, 0, &Connection);
00745     if (Result!=ZESTETM1_SUCCESS)
00746     {
00747         ZESTETM1_ERROR("ZestETM1ReadRegister", Result);
00748     }
00749 }

```

```

00750     // Read register from the device
00751     Cmd.Command = ZESTETM1_COMMAND_READ_REG;
00752     Cmd.Addr = (uint8_t)Addr;
00753     Result = ZestETM1_SendCommand(CardInfo, Connection,
00754                                   &Cmd, sizeof(Cmd),
00755                                   &Response, sizeof(Response), 1);
00756     if (Result!=ZESTETM1_SUCCESS)
00757     {
00758         ZestETM1_CloseConnection(Connection);
00759         ZESTETM1_ERROR("ZestETM1ReadRegister", Result);
00760     }
00761     if (Response.Command!=Cmd.Command || Response.Status!=0)
00762     {
00763         ZestETM1_CloseConnection(Connection);
00764         ZESTETM1_ERROR("ZestETM1ReadRegister", ZESTETM1_INTERNAL_ERROR);
00765     }
00766     *Data = ((Response.Value>>8)&0xff) | ((Response.Value&0xff)<<8);
00767
00768     Result = ZestETM1_CloseConnection(Connection);
00769     if (Result!=ZESTETM1_SUCCESS)
00770     {
00771         ZESTETM1_ERROR("ZestETM1ReadRegister", Result);
00772     }
00773
00774     return ZESTETM1_SUCCESS;
00775 }

```

7.31.5.11 ZestETM1RegisterErrorHandler()

```

ZESTETM1_STATUS ZestETM1RegisterErrorHandler (
    ZESTETM1_ERROR_FUNC Function )

```

Definition at line 74 of file [Error.c](#).

```

00075 {
00076     ZestETM1_ErrorHandler = Function;
00077     return ZESTETM1_SUCCESS;
00078 }

```

7.31.5.12 ZestETM1SetInterrupt()

```

ZESTETM1_STATUS ZestETM1SetInterrupt (
    ZESTETM1_CARD_INFO * CardInfo )

```

Definition at line 780 of file [Data.c](#).

```

00781 {
00782     ZESTETM1_STATUS Result;
00783     ZESTETM1_CONNECTION Connection;
00784     ZESTETM1_MAILBOX_INT_CMD Cmd;
00785     ZESTETM1_MAILBOX_INT_RESPONSE Response;
00786
00787     if (CardInfo==NULL)
00788     {
00789         ZESTETM1_ERROR("ZestETM1SetInterrupt", ZESTETM1_NULL_PARAMETER);
00790     }
00791
00792     Result = ZestETM1_OpenConnection(CardInfo, ZESTETM1_TYPE_TCP, CardInfo->ControlPort, 0, &Connection);
00793     if (Result!=ZESTETM1_SUCCESS)
00794     {
00795         ZESTETM1_ERROR("ZestETM1SetInterrupt", Result);
00796     }
00797
00798     // Send command to set interrupt
00799     Cmd.Command = ZESTETM1_COMMAND_MAILBOX_INT;
00800     Result = ZestETM1_SendCommand(CardInfo, Connection,
00801                                   &Cmd, sizeof(Cmd),
00802                                   &Response, sizeof(Response), 1);
00803     if (Result!=ZESTETM1_SUCCESS)
00804     {
00805         ZestETM1_CloseConnection(Connection);

```

```

00806         ZESTETM1_ERROR("ZestETM1SetInterrupt", Result);
00807     }
00808     if (Response.Command!=Cmd.Command || Response.Status!=0)
00809     {
00810         ZestETM1_CloseConnection(Connection);
00811         ZESTETM1_ERROR("ZestETM1SetInterrupt", ZESTETM1_INTERNAL_ERROR);
00812     }
00813
00814     Result = ZestETM1_CloseConnection(Connection);
00815     if (Result!=ZESTETM1_SUCCESS)
00816     {
00817         ZESTETM1_ERROR("ZestETM1SetInterrupt", Result);
00818     }
00819
00820     return ZESTETM1_SUCCESS;
00821 }

```

7.31.5.13 ZestETM1SPIReadWrite()

```

ZESTETM1_STATUS ZestETM1SPIReadWrite (
    ZESTETM1_CARD_INFO * CardInfo,
    ZESTETM1_SPI_RATE Rate,
    int WordLen,
    void * WriteData,
    void * ReadData,
    unsigned long Length,
    int ReleaseCS )

```

Definition at line 635 of file [Data.c](#).

```

00639 {
00640     ZESTETM1_STATUS Result;
00641     ZESTETM1_CONNECTION Connection;
00642     unsigned long RateVal = Rate==ZESTETM1_SPI_RATE_35 ? ZESTETM1_RATE_40MHz :
00643         Rate==ZESTETM1_SPI_RATE_17_5 ? ZESTETM1_RATE_20MHz : ZESTETM1_RATE_10MHz;
00644
00645     if (CardInfo==NULL || (WriteData==NULL && ReadData==NULL))
00646     {
00647         ZESTETM1_ERROR("ZestETM1SPIReadWrite", ZESTETM1_NULL_PARAMETER);
00648     }
00649     if (WordLen<1 || WordLen>32 || Length>16384)
00650     {
00651         ZESTETM1_ERROR("ZestETM1SPIReadWrite", ZESTETM1_ILLEGAL_PARAMETER);
00652     }
00653
00654     Result = ZestETM1_OpenConnection(CardInfo, ZESTETM1_TYPE_TCP, CardInfo->ControlPort, 0, &Connection);
00655     if (Result!=ZESTETM1_SUCCESS)
00656     {
00657         ZESTETM1_ERROR("ZestETM1SPIReadWrite", Result);
00658     }
00659
00660     Result = ZestETM1_SPIReadWrite(CardInfo, Connection, ZESTETM1_USER_DEVICE_ID|RateVal, WordLen,
    (uint32_t *)WriteData, (uint32_t *)ReadData, Length, ReleaseCS, 1);
00661     if (Result!=ZESTETM1_SUCCESS)
00662     {
00663         ZESTETM1_ERROR("ZestETM1SPIReadWrite", Result);
00664     }
00665
00666     Result = ZestETM1_CloseConnection(Connection);
00667     if (Result!=ZESTETM1_SUCCESS)
00668     {
00669         ZESTETM1_ERROR("ZestETM1SPIReadWrite", Result);
00670     }
00671
00672     return ZESTETM1_SUCCESS;
00673 }

```

7.31.5.14 ZestETM1WriteData()

```

ZESTETM1_STATUS ZestETM1WriteData (
    ZESTETM1_CONNECTION Connection,

```

```

    void * Buffer,
    unsigned long Length,
    unsigned long * Written,
    unsigned long Timeout )

```

Definition at line 592 of file [Data.c](#).

```

00597 {
00598     ZESTETM1_STATUS Result;
00599     ZESTETM1_CONNECTION_STRUCT *Conn = (ZESTETM1_CONNECTION_STRUCT*)Connection;
00600
00601     Result = ZestETM1_WriteData(Connection, Buffer, Length, Written, Timeout);
00602     if (Result!=ZESTETM1_SUCCESS)
00603     {
00604         ZESTETM1_ERROR_CONN("ZestETM1WriteData", Result);
00605     }
00606
00607     return ZESTETM1_SUCCESS;
00608 }

```

7.31.5.15 ZestETM1WriteRegister()

```

ZESTETM1_STATUS ZestETM1WriteRegister (
    ZESTETM1_CARD_INFO * CardInfo,
    unsigned long Addr,
    unsigned short Data )

```

Definition at line 678 of file [Data.c](#).

```

00679 {
00680     ZESTETM1_STATUS Result;
00681     ZESTETM1_CONNECTION Connection;
00682     ZESTETM1_WRITE_REG_CMD Cmd;
00683     ZESTETM1_WRITE_REG_RESPONSE Response;
00684
00685     if (CardInfo==NULL)
00686     {
00687         ZESTETM1_ERROR("ZestETM1WriteRegister", ZESTETM1_NULL_PARAMETER);
00688     }
00689     if (Addr>127)
00690     {
00691         ZESTETM1_ERROR("ZestETM1WriteRegister", ZESTETM1_ILLEGAL_PARAMETER);
00692     }
00693
00694     Result = ZestETM1_OpenConnection(CardInfo, ZESTETM1_TYPE_TCP, CardInfo->ControlPort, 0, &Connection);
00695     if (Result!=ZESTETM1_SUCCESS)
00696     {
00697         ZESTETM1_ERROR("ZestETM1WriteRegister", Result);
00698     }
00699
00700     // Write register to the device
00701     //FIXME: Do we want to be able to copy this value to flash?
00702     Cmd.Command = ZESTETM1_COMMAND_WRITE_REG;
00703     Cmd.Addr = (uint8_t)Addr;
00704     Cmd.Data = ((Data>>8)&0xff) | (Data&0xff);
00705     Result = ZestETM1_SendCommand(CardInfo, Connection,
00706                                   &Cmd, sizeof(Cmd),
00707                                   &Response, sizeof(Response), 1);
00708     if (Result!=ZESTETM1_SUCCESS)
00709     {
00710         ZestETM1_CloseConnection(Connection);
00711         ZESTETM1_ERROR("ZestETM1WriteRegister", Result);
00712     }
00713     if (Response.Command!=Cmd.Command || Response.Status!=0)
00714     {
00715         ZestETM1_CloseConnection(Connection);
00716         ZESTETM1_ERROR("ZestETM1WriteRegister", ZESTETM1_INTERNAL_ERROR);
00717     }
00718
00719     Result = ZestETM1_CloseConnection(Connection);
00720     if (Result!=ZESTETM1_SUCCESS)
00721     {
00722         ZESTETM1_ERROR("ZestETM1WriteRegister", Result);
00723     }
00724
00725     return ZESTETM1_SUCCESS;
00726 }

```

7.32 ZestETM1.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002 *
00003 * (c) 2012 Orange Tree Technologies Ltd
00004 *
00005 * ZestETM1.h
00006 * Version 1.0
00007 *
00008 * Header file for ZestETM1 Ethernet module
00009 *
00010 *****/
00011
00012 #ifndef __ZESTETM1_H__
00013 #define __ZESTETM1_H__
00014
00015 #ifdef __cplusplus
00016 extern "C"
00017 {
00018 #endif
00019
00020
00021 /*****
00022 * Handle for referencing modules *
00023 *****/
00024 typedef void *ZESTETM1_HANDLE;
00025
00026
00027 /*****
00028 * Card information structure *
00029 *****/
00030 typedef struct
00031 {
00032     // These must be filled in before calling functions
00033     unsigned char IPAddr[4];
00034     unsigned short ControlPort;
00035     unsigned long Timeout;
00036
00037     // These are for information purposes only
00038     unsigned short HTTPPort;
00039     unsigned char MACAddr[6];
00040     unsigned char SubNet[4];
00041     unsigned char Gateway[4];
00042     unsigned long SerialNumber;
00043     unsigned long FirmwareVersion;
00044     unsigned long HardwareVersion;
00045 } ZESTETM1_CARD_INFO;
00046
00047 // Fallback mask
00048 // This will be set if the GigExpedite is operating in firmware version fallback mode
00049 // due to a failed upload of firmware
00050 #define ZESTETM1_VERSION_FALLBACK 0x8000
00051
00052 /*****
00053 * Data transfer definitions *
00054 *****/
00055 typedef void *ZESTETM1_CONNECTION;
00056 typedef enum
00057 {
00058     ZESTETM1_TYPE_TCP,
00059     ZESTETM1_TYPE_UDP,
00060 } ZESTETM1_CONNECTION_TYPE;
00061
00062 /*****
00063 * Master SPI clock rates *
00064 *****/
00065 typedef enum
00066 {
00067     ZESTETM1_SPI_RATE_35,
00068     ZESTETM1_SPI_RATE_17_5,
00069     ZESTETM1_SPI_RATE_8_75,
00070 } ZESTETM1_SPI_RATE;
00071
00072 /*****
00073 * Function return codes *
00074 *****/
00075 #define ZESTETM1_INFO_BASE 0
00076 #define ZESTETM1_WARNING_BASE 0x4000

```

```

00077 #define ZESTETM1_ERROR_BASE 0x8000
00078 typedef enum
00079 {
00080     ZESTETM1_SUCCESS = ZESTETM1_INFO_BASE,
00081     ZESTETM1_MAX_INFO,
00082
00083     ZESTETM1_MAX_WARNING = ZESTETM1_WARNING_BASE,
00084
00085     ZESTETM1_SOCKET_ERROR = ZESTETM1_ERROR_BASE,
00086     ZESTETM1_INTERNAL_ERROR,
00087     ZESTETM1_ILLEGAL_STATUS_CODE,
00088     ZESTETM1_NULL_PARAMETER,
00089     ZESTETM1_OUT_OF_MEMORY,
00090     ZESTETM1_INVALID_CONNECTION_TYPE,
00091     ZESTETM1_ILLEGAL_CONNECTION,
00092     ZESTETM1_SOCKET_CLOSED,
00093     ZESTETM1_TIMEOUT,
00094     ZESTETM1_ILLEGAL_PARAMETER,
00095
00096     ZESTETM1_MAX_ERROR
00097 } ZESTETM1_STATUS;
00098 typedef void (*ZESTETM1_ERROR_FUNC)(const char *Function,
00099                                     ZESTETM1_CARD_INFO *CardInfo,
00100                                     ZESTETM1_STATUS Status,
00101                                     const char *Msg);
00102
00103
00104 /*****
00105  * Function prototypes *
00106  *****/
00107 ZESTETM1_STATUS ZestETM1Init(void);
00108 ZESTETM1_STATUS ZestETM1Close(void);
00109 ZESTETM1_STATUS ZestETM1CountCards(unsigned long *NumCards,
00110                                     ZESTETM1_CARD_INFO **CardInfo,
00111                                     unsigned long Wait);
00112 ZESTETM1_STATUS ZestETM1GetCardInfo(ZESTETM1_CARD_INFO *CardInfo);
00113 ZESTETM1_STATUS ZestETM1FreeCards(ZESTETM1_CARD_INFO *CardInfo);
00114
00115 ZESTETM1_STATUS ZestETM1RegisterErrorHandler(ZESTETM1_ERROR_FUNC Function);
00116 ZESTETM1_STATUS ZestETM1GetErrorMessage(ZESTETM1_STATUS Status,
00117                                          char **Buffer);
00118
00119 ZESTETM1_STATUS ZestETM1OpenConnection(ZESTETM1_CARD_INFO *CardInfo,
00120                                         ZESTETM1_CONNECTION_TYPE Type,
00121                                         unsigned short Port,
00122                                         unsigned short LocalPort,
00123                                         ZESTETM1_CONNECTION *Connection);
00124 ZESTETM1_STATUS ZestETM1CloseConnection(ZESTETM1_CONNECTION Connection);
00125 ZESTETM1_STATUS ZestETM1WriteData(ZESTETM1_CONNECTION Connection,
00126                                   void *Buffer,
00127                                   unsigned long Length,
00128                                   unsigned long *Written,
00129                                   unsigned long Timeout);
00130 ZESTETM1_STATUS ZestETM1ReadData(ZESTETM1_CONNECTION Connection,
00131                                  void *Buffer,
00132                                  unsigned long Length,
00133                                  unsigned long *Read,
00134                                  unsigned long Timeout);
00135
00136 ZESTETM1_STATUS ZestETM1SPIReadWrite(ZESTETM1_CARD_INFO *CardInfo, ZESTETM1_SPI_RATE Rate,
00137                                       int WordLen, void *WriteData,
00138                                       void *ReadData, unsigned long Length,
00139                                       int ReleaseCS);
00140
00141 ZESTETM1_STATUS ZestETM1WriteRegister(ZESTETM1_CARD_INFO *CardInfo, unsigned long Addr, unsigned short
00142 Data);
00143 ZESTETM1_STATUS ZestETM1ReadRegister(ZESTETM1_CARD_INFO *CardInfo, unsigned long Addr, unsigned short
00144 *Data);
00145 ZESTETM1_STATUS ZestETM1SetInterrupt(ZESTETM1_CARD_INFO *CardInfo);
00146
00147 #ifdef __cplusplus
00148 }
00149 #endif
00150 #endif // __ZESTETM1_H__

```


7.33 C:/Users/hill35/git/camera_python/nsCamera/sensors/daedalus.py File Reference

Classes

- class [nsCamera.sensors.daedalus.daedalus](#)

Namespaces

- namespace [nsCamera](#)
- namespace [nsCamera.sensors](#)
- namespace [nsCamera.sensors.daedalus](#)

7.34 daedalus.py

[Go to the documentation of this file.](#)

```

00001 # -*- coding: utf-8 -*-
00002 """
00003 Parameters and functions specific to the daedalus three-frame sensor
00004
00005
00006 Author: Jeremy Martin Hill (jerhill@llnl.gov)
00007
00008 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00009 LLNL-CODE-838080
00010
00011 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00012 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00013 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00014 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00015 be made under this license.
00016
00017 Version: 2.1.2 (February 2025)
00018 """
00019
00020 import logging
00021 import numbers
00022 from collections import OrderedDict
00023
00024 import numpy as np
00025
00026 from nsCamera.sensors.sensorBase import sensorBase
00027 from nsCamera.utils.misc import flattenlist
00028
00029
00030 class daedalus(sensorBase):
00031     specwarn = ""
00032     minframe = 0 # fixed value for sensor
00033     maxframe = 2 # fixed value for sensor
00034     maxwidth = 512 # fixed value for sensor
00035     maxheight = 1024 # fixed value for sensor
00036     bytesperpixel = 2
00037     fpganumID = 2 # last nybble of FPGA_NUM
00038     detect = "DAEDALUS_DET"
00039     sensfam = "Daedalus"
00040     loglabel = "[Daedalus] "
00041     ZDT = False
00042     HFW = False
00043     firstframe = 0
00044     lastframe = 2
00045     nframes = 3
00046     width = 512
00047     height = 1024
00048     firstrow = 0
00049     lastrow = 1023

```

```

00050     interlacing = [0, 0]
00051     columns = 1
00052     padToFull = True
00053     toffset = -165.76 # default temperature sensor offset
00054     tslope = 81.36 # default temperature sensor slope
00055
00056     def __init__(self, ca):
00057         self.caca = ca
00058         super(daedalus, self).__init__(ca)
00059
00060         self.sens_registers = OrderedDict(
00061             {
00062                 "HST_READBACK_A_LO": "018",
00063                 "HST_READBACK_A_HI": "019",
00064                 "HST_READBACK_B_LO": "01A",
00065                 "HST_READBACK_B_HI": "01B",
00066                 "HSTALLWEN_WAIT_TIME": "03F",
00067                 "VRESET_HIGH_VALUE": "04A",
00068                 "FRAME_ORDER_SEL": "04B",
00069                 "EXT_PHI_CLK_SH0_ON": "050",
00070                 "EXT_PHI_CLK_SH0_OFF": "051",
00071                 "EXT_PHI_CLK_SH1_ON": "052",
00072                 "EXT_PHI_CLK_SH1_OFF": "053",
00073                 "EXT_PHI_CLK_SH2_ON": "054",
00074                 "HST_TRIGGER_DELAY_DATA_LO": "120",
00075                 "HST_TRIGGER_DELAY_DATA_HI": "121",
00076                 "HST_PHI_DELAY_DATA": "122",
00077                 "HST_EXT_CLK_HALF_PER": "129",
00078                 "HST_COUNT_TRIG": "130",
00079                 "HST_DELAY_EN": "131",
00080                 "RSL_HFW_MODE_EN": "133",
00081                 "RSL_ZDT_MODE_B_EN": "135",
00082                 "RSL_ZDT_MODE_A_EN": "136",
00083                 "BGTRIMA": "137",
00084                 "BGTRIMB": "138",
00085                 "COLUMN_TEST_EN": "139",
00086                 "RSL_CONFIG_DATA_B0": "140",
00087                 "RSL_CONFIG_DATA_B1": "141",
00088                 "RSL_CONFIG_DATA_B2": "142",
00089                 "RSL_CONFIG_DATA_B3": "143",
00090                 "RSL_CONFIG_DATA_B4": "144",
00091                 "RSL_CONFIG_DATA_B5": "145",
00092                 "RSL_CONFIG_DATA_B6": "146",
00093                 "RSL_CONFIG_DATA_B7": "147",
00094                 "RSL_CONFIG_DATA_B8": "148",
00095                 "RSL_CONFIG_DATA_B9": "149",
00096                 "RSL_CONFIG_DATA_B10": "14A",
00097                 "RSL_CONFIG_DATA_B11": "14B",
00098                 "RSL_CONFIG_DATA_B12": "14C",
00099                 "RSL_CONFIG_DATA_B13": "14D",
00100                 "RSL_CONFIG_DATA_B14": "14E",
00101                 "RSL_CONFIG_DATA_B15": "14F",
00102                 "RSL_CONFIG_DATA_B16": "150",
00103                 "RSL_CONFIG_DATA_B17": "151",
00104                 "RSL_CONFIG_DATA_B18": "152",
00105                 "RSL_CONFIG_DATA_B19": "153",
00106                 "RSL_CONFIG_DATA_B20": "154",
00107                 "RSL_CONFIG_DATA_B21": "155",
00108                 "RSL_CONFIG_DATA_B22": "156",
00109                 "RSL_CONFIG_DATA_B23": "157",
00110                 "RSL_CONFIG_DATA_B24": "158",
00111                 "RSL_CONFIG_DATA_B25": "159",
00112                 "RSL_CONFIG_DATA_B26": "15A",
00113                 "RSL_CONFIG_DATA_B27": "15B",
00114                 "RSL_CONFIG_DATA_B28": "15C",
00115                 "RSL_CONFIG_DATA_B29": "15D",
00116                 "RSL_CONFIG_DATA_B30": "15E",
00117                 "RSL_CONFIG_DATA_B31": "15F",
00118                 "RSL_CONFIG_DATA_A0": "160",
00119                 "RSL_CONFIG_DATA_A1": "161",
00120                 "RSL_CONFIG_DATA_A2": "162",
00121                 "RSL_CONFIG_DATA_A3": "163",
00122                 "RSL_CONFIG_DATA_A4": "164",
00123                 "RSL_CONFIG_DATA_A5": "165",
00124                 "RSL_CONFIG_DATA_A6": "166",
00125                 "RSL_CONFIG_DATA_A7": "167",
00126                 "RSL_CONFIG_DATA_A8": "168",
00127                 "RSL_CONFIG_DATA_A9": "169",
00128                 "RSL_CONFIG_DATA_A10": "16A",
00129                 "RSL_CONFIG_DATA_A11": "16B",
00130                 "RSL_CONFIG_DATA_A12": "16C",

```

```

00131         "RSL_CONFIG_DATA_A13": "16D",
00132         "RSL_CONFIG_DATA_A14": "16E",
00133         "RSL_CONFIG_DATA_A15": "16F",
00134         "RSL_CONFIG_DATA_A16": "170",
00135         "RSL_CONFIG_DATA_A17": "171",
00136         "RSL_CONFIG_DATA_A18": "172",
00137         "RSL_CONFIG_DATA_A19": "173",
00138         "RSL_CONFIG_DATA_A20": "174",
00139         "RSL_CONFIG_DATA_A21": "175",
00140         "RSL_CONFIG_DATA_A22": "176",
00141         "RSL_CONFIG_DATA_A23": "177",
00142         "RSL_CONFIG_DATA_A24": "178",
00143         "RSL_CONFIG_DATA_A25": "179",
00144         "RSL_CONFIG_DATA_A26": "17A",
00145         "RSL_CONFIG_DATA_A27": "17B",
00146         "RSL_CONFIG_DATA_A28": "17C",
00147         "RSL_CONFIG_DATA_A29": "17D",
00148         "RSL_CONFIG_DATA_A30": "17E",
00149         "RSL_CONFIG_DATA_A31": "17F",
00150     }
00151 )
00152
00153 self.sens_subregisters = [
00154
00155     ("HST_MODE", "HS_TIMING_CTL", 0, 1, True),
00156     ("SLOWREADOFF_0", "CTRL_REG", 4, 1, True),
00157     ("SLOWREADOFF_1", "CTRL_REG", 5, 1, True),
00158     ("MANSHUT_MODE", "CTRL_REG", 8, 1, True),
00159     ("INTERLACING_EN", "CTRL_REG", 9, 1, True),
00160     ("HFW", "RSL_HFW_MODE_EN", 0, 1, True),
00161     ("ZDT_A", "RSL_ZDT_MODE_A_EN", 0, 1, True),
00162     ("ZDT_B", "RSL_ZDT_MODE_B_EN", 0, 1, True),
00163     ("HST_DEL_EN", "HST_DELAY_EN", 0, 1, True),
00164     ("PHI_DELAY_A", "HST_PHI_DELAY_DATA", 9, 10, True),
00165     ("PHI_DELAY_B", "HST_PHI_DELAY_DATA", 29, 10, True),
00166     # Assume that daedalus is not to be used with v1 board
00167     ("VRESET_HIGH", "VRESET_HIGH_VALUE", 15, 16, True),
00168
00169     ("STAT_SHORISEUR", "STAT_REG", 3, 1, False),
00170     ("STAT_SHOFALLUR", "STAT_REG", 4, 1, False),
00171     ("STAT_RSLNALLWENA", "STAT_REG", 12, 1, False),
00172     ("STAT_RSLNALLWENB", "STAT_REG", 15, 1, False),
00173     # ("STAT_CONFIGSTDONE", "STAT_REG", 16, 1, False),
00174 ]
00175
00176 # TODO: add warning if daedalus and v1 board are together
00177 def sensorSpecific(self):
00178     """
00179     Returns:
00180         list of tuples, (Sensor-specific register, default setting)
00181     """
00182     return [
00183         ("FPA_FRAME_INITIAL", "00000000"),
00184         ("FPA_FRAME_FINAL", "00000002"),
00185         ("FPA_ROW_INITIAL", "00000000"),
00186         ("FPA_ROW_FINAL", "000003FF"),
00187         ("HS_TIMING_DATA_ALO", "00006666", # 0db6 = 2-1; 6666 = 2-2
00188         ("HS_TIMING_DATA_AHI", "00000000"),
00189         ("HS_TIMING_DATA_BLO", "00006666"),
00190         ("HS_TIMING_DATA_BHI", "00000000"),
00191         ("FRAME_ORDER_SEL", "00000000"),
00192         ("RSL_HFW_MODE_EN", "00000000"),
00193         ("RSL_ZDT_MODE_B_EN", "00000000"),
00194         ("RSL_ZDT_MODE_A_EN", "00000000"),
00195         ("RSL_CONFIG_DATA_B0", "00000000"),
00196         ("RSL_CONFIG_DATA_B1", "00000000"),
00197         ("RSL_CONFIG_DATA_B2", "00000000"),
00198         ("RSL_CONFIG_DATA_B3", "00000000"),
00199         ("RSL_CONFIG_DATA_B4", "00000000"),
00200         ("RSL_CONFIG_DATA_B5", "00000000"),
00201         ("RSL_CONFIG_DATA_B6", "00000000"),
00202         ("RSL_CONFIG_DATA_B7", "00000000"),
00203         ("RSL_CONFIG_DATA_B8", "00000000"),
00204         ("RSL_CONFIG_DATA_B9", "00000000"),
00205         ("RSL_CONFIG_DATA_B10", "00000000"),
00206         ("RSL_CONFIG_DATA_B11", "00000000"),
00207         ("RSL_CONFIG_DATA_B12", "00000000"),
00208         ("RSL_CONFIG_DATA_B13", "00000000"),
00209         ("RSL_CONFIG_DATA_B14", "00000000"),
00210         ("RSL_CONFIG_DATA_B15", "00000000"),
00211         ("RSL_CONFIG_DATA_B16", "00000000"),

```

```

00215         ("RSL_CONFIG_DATA_B17", "00000000"),
00216         ("RSL_CONFIG_DATA_B18", "00000000"),
00217         ("RSL_CONFIG_DATA_B19", "00000000"),
00218         ("RSL_CONFIG_DATA_B20", "00000000"),
00219         ("RSL_CONFIG_DATA_B21", "00000000"),
00220         ("RSL_CONFIG_DATA_B22", "00000000"),
00221         ("RSL_CONFIG_DATA_B23", "00000000"),
00222         ("RSL_CONFIG_DATA_B24", "00000000"),
00223         ("RSL_CONFIG_DATA_B25", "00000000"),
00224         ("RSL_CONFIG_DATA_B26", "00000000"),
00225         ("RSL_CONFIG_DATA_B27", "00000000"),
00226         ("RSL_CONFIG_DATA_B28", "00000000"),
00227         ("RSL_CONFIG_DATA_B29", "00000000"),
00228         ("RSL_CONFIG_DATA_B30", "00000000"),
00229         ("RSL_CONFIG_DATA_B31", "00000000"),
00230         ("RSL_CONFIG_DATA_A0", "00000000"),
00231         ("RSL_CONFIG_DATA_A1", "00000000"),
00232         ("RSL_CONFIG_DATA_A2", "00000000"),
00233         ("RSL_CONFIG_DATA_A3", "00000000"),
00234         ("RSL_CONFIG_DATA_A4", "00000000"),
00235         ("RSL_CONFIG_DATA_A5", "00000000"),
00236         ("RSL_CONFIG_DATA_A6", "00000000"),
00237         ("RSL_CONFIG_DATA_A7", "00000000"),
00238         ("RSL_CONFIG_DATA_A8", "00000000"),
00239         ("RSL_CONFIG_DATA_A9", "00000000"),
00240         ("RSL_CONFIG_DATA_A10", "00000000"),
00241         ("RSL_CONFIG_DATA_A11", "00000000"),
00242         ("RSL_CONFIG_DATA_A12", "00000000"),
00243         ("RSL_CONFIG_DATA_A13", "00000000"),
00244         ("RSL_CONFIG_DATA_A14", "00000000"),
00245         ("RSL_CONFIG_DATA_A15", "00000000"),
00246         ("RSL_CONFIG_DATA_A16", "00000000"),
00247         ("RSL_CONFIG_DATA_A17", "00000000"),
00248         ("RSL_CONFIG_DATA_A18", "00000000"),
00249         ("RSL_CONFIG_DATA_A19", "00000000"),
00250         ("RSL_CONFIG_DATA_A20", "00000000"),
00251         ("RSL_CONFIG_DATA_A21", "00000000"),
00252         ("RSL_CONFIG_DATA_A22", "00000000"),
00253         ("RSL_CONFIG_DATA_A23", "00000000"),
00254         ("RSL_CONFIG_DATA_A24", "00000000"),
00255         ("RSL_CONFIG_DATA_A25", "00000000"),
00256         ("RSL_CONFIG_DATA_A26", "00000000"),
00257         ("RSL_CONFIG_DATA_A27", "00000000"),
00258         ("RSL_CONFIG_DATA_A28", "00000000"),
00259         ("RSL_CONFIG_DATA_A29", "00000000"),
00260         ("RSL_CONFIG_DATA_A30", "00000000"),
00261         ("RSL_CONFIG_DATA_A31", "00000000"),
00262         ("HST_TRIGGER_DELAY_DATA_LO", "00000000"),
00263         ("HST_TRIGGER_DELAY_DATA_HI", "00000000"),
00264         ("HST_PHI_DELAY_DATA", "00000000"),
00265         ("SLOWREADOFF_0", "0"),
00266         ("SLOWREADOFF_1", "0"),
00267     ]
00268
00269     def setInterlacing(self, ifactor=None, side=None):
00270         """
00271         Sets interlacing factor. NOTE: if called directly when HFW or ZDT mode is active,
00272         this will disengage those modes automatically. If hemispheres have different
00273         factors when the image is acquired, the resulting frames are separated into
00274         half-width images
00275
00276         Args:
00277             ifactor: number of interlaced lines (generates ifactor + 1 images per frame)
00278                     defaults to 0 (no interlacing)
00279             side: identify particular hemisphere (A or B) to control. If left blank,
00280                  control both hemispheres
00281
00282         Returns:
00283             integer: active interlacing factor (unchanged if error)
00284         """
00285         logging.debug(self.logdebug + "setInterlacing; ifactor = " + str(ifactor))
00286         if ifactor is None:
00287             ifactor = 0
00288         if (
00289             not isinstance(ifactor, int)
00290             or ifactor < 0
00291             or ifactor > (self.maxheight - 1)
00292         ):
00293             err = (
00294                 self.logerr + "invalid interlacing factor submitted. "
00295                 "Interlacing remains unchanged. "

```

```

00296         )
00297         logging.error(err)
00298         return self.interlacinginterlacing
00299     if self.HFWHFW:
00300         logging.warning(
00301             self.logwarn + "HFW mode will be disengaged because of new "
00302             "interlacing setting "
00303         )
00304         self.setHighFullWellsetHighFullWell(False)
00305     if self.ZDTZDT:
00306         logging.warning(
00307             self.logwarn + "ZDT mode will be disengaged because of new "
00308             "interlacing setting "
00309         )
00310         self.setZeroDeadTimesetZeroDeadTime(False)
00311     if ifactor == 0:
00312         bitscheme = self.maxheight * [0]
00313         # deactivating one side shouldn't turn off enable for both sides
00314         # TODO: is it a problem if sides are set separately, so interlacing is zero
00315         # but still enabled?
00316         if side is None:
00317             self.caca.setSubregister("INTERLACING_EN", "0")
00318     else:
00319         pattern = [0] + ifactor * [1]
00320         reps = 1 + self.maxheight // (ifactor + 1)
00321         bitscheme = (reps * pattern)[0 : self.maxheight]
00322         self.caca.setSubregister("INTERLACING_EN", "1")
00323     err = ""
00324     for regnum in range(32):
00325         regbits = bitscheme[32 * regnum : 32 * (regnum + 1)]
00326         logging.debug(self.logdebug + "regbits = " + str(regbits))
00327         # generated pattern is reverse order from placement in register (element 0
00328         # of the list is the LSB of the register)
00329         bitsrev = regbits[::-1]
00330         s = [str(i) for i in bitsrev]
00331         b = "".join(s) # assemble as binary number for processing
00332         hexval = "%x" % int(b, 2)
00333         val = hexval.zfill(8)
00334         err0 = ""
00335         err1 = ""
00336         if side is None or side.lower() == "a":
00337             lname = "RSL_CONFIG_DATA_A" + str(regnum)
00338             err1, _ = self.caca.setRegister(lname, val)
00339             self.interlacinginterlacing[1] = ifactor
00340         if side is None or side.lower() == "b":
00341             rname = "RSL_CONFIG_DATA_B" + str(regnum)
00342             err0, _ = self.caca.setRegister(rname, val)
00343             self.interlacinginterlacing[0] = ifactor
00344         err = err + err0 + err1
00345     if err:
00346         logging.error(self.logerr + "interlacing may not be set correctly: " + err)
00347     logging.info(self.loginfo + "Interlacing set to " + str(self.interlacinginterlacing))
00348     if self.interlacinginterlacing[0] == self.interlacinginterlacing[1]:
00349         self.columnscolumns = 1
00350     else:
00351         self.columnscolumns = 2
00352     return self.interlacinginterlacing
00353
00354 def setHighFullWell(self, flag):
00355     """
00356     Activates High Full Well mode. All frames are acquired simultaneously. Zero Dead
00357     Time mode and interlacing will be automatically deactivated and column number
00358     will be reset to 0. NOTE: after deactivating HFW, the board remains in
00359     uninterlaced mode (interlacing = 0)
00360
00361     Args:
00362         flag: True to activate HFW mode, False to deactivate
00363
00364     Returns:
00365         Error message
00366     """
00367     logging.debug(self.logdebug + "setHighFullWell; flag = " + str(flag))
00368     err0 = ""
00369     if flag:
00370         if self.ZDTZDT:
00371             logging.warning(
00372                 self.logwarn + "ZDT mode will be disengaged because of HFW "
00373                 "setting "
00374             )
00375             err0 = self.setZeroDeadTimesetZeroDeadTime(False)
00376             err1, _ = self.caca.setSubregister("HFW", "1")

```

```

00377         self.HFWHFW = False # preclude HFW deactivation message in setInterlacing
00378         self.setInterlacingsetInterlacing(0)
00379         self.HFWHFW = True
00380         logging.info(self.loginfo + "High Full Well mode active")
00381     else:
00382         self.HFWHFW = False
00383         err1, _ = self.caca.setSubregister("HFW", "0")
00384         logging.info(self.loginfo + "High Full Well mode inactivate")
00385     err = err0 + err1
00386     if err:
00387         logging.error(self.logerr + "HFW option may not be set correctly ")
00388     return err
00389
00390 def setZeroDeadTime(self, flag=True, side=None):
00391     """
00392     Activates Zero Dead Time mode. Even rows follow the assigned HST schedule; odd
00393     rows are acquired while the 'shutter' for the even rows are closed. High Full
00394     Well mode and interlacing will be automatically deactivated.
00395     *NOTE* after deactivating ZDT, the board reverts to uninterlaced mode
00396     (interlacing = 0)
00397
00398     Args:
00399         flag: True to activate ZDT mode, False to deactivate
00400         side: identify particular hemisphere (A or B) to control. If left blank,
00401             control both hemispheres
00402
00403     Returns:
00404         Error message
00405     """
00406     logging.debug(self.logdebug + "setZeroDeadTime; flag = " + str(flag))
00407     err0 = ""
00408     err1 = ""
00409     err2 = ""
00410     if flag:
00411         if self.HFWHFW:
00412             logging.warning(
00413                 self.logwarn + "HFW mode will be disengaged because of ZDT "
00414                 "setting "
00415             )
00416             err0 = self.setHighFullWellsetHighFullWell(False)
00417             if side is None or side.lower() == "a":
00418                 err2, _ = self.caca.setSubregister("ZDT_A", "1")
00419                 self.interlacinginterlacing[0] = 1
00420             if side is None or side.lower() == "b":
00421                 err1, _ = self.caca.setSubregister("ZDT_B", "1")
00422                 self.interlacinginterlacing[1] = 1
00423             # self.ZDT = False # preclude ZDT deactivation message in setInterlacing
00424             # if self.interlacing != [0, 0]:
00425             #     self.setInterlacing(0)
00426             # TODO: need to handle flags when ZDT active for just one side
00427             self.ZDTZDT = True
00428             logging.info(
00429                 self.loginfo + "Zero Dead Time mode active; actual interlacing = 1"
00430             )
00431         else:
00432             self.ZDTZDT = False
00433             if side is None or side.lower() == "a":
00434                 err2, _ = self.caca.setSubregister("ZDT_A", "0")
00435             if side is None or side.lower() == "b":
00436                 err1, _ = self.caca.setSubregister("ZDT_B", "0")
00437             self.setInterlacingsetInterlacing(0)
00438             logging.info(self.loginfo + "Zero Dead Time mode inactivate")
00439     err = err0 + err1 + err2
00440     if err:
00441         logging.error(self.logerr + "ZDT option may not be set correctly ")
00442     return err
00443
00444 def selectOscillator(self, osc=None):
00445     """
00446     Selects oscillator to control sensor timing
00447     Args:
00448         osc: 500|100|'ring'|'external', defaults to 500 MHz
00449
00450     Returns:
00451         error message as string
00452     """
00453     logging.info(self.loginfo + "selectOscillator; osc = " + str(osc))
00454     if osc is None:
00455         osc = 500
00456     osc = str(osc)
00457     if osc[:3] == "500":

```

```

00458         payload = "00"
00459     elif osc[:3] == "100":
00460         payload = "01"
00461     elif osc.upper()[3] == "RIN":
00462         payload = "10"
00463     elif osc.upper()[3] in ["EXT"]:
00464         payload = "11"
00465     else:
00466         err = (
00467             self.logerr + "selectOscillator: invalid parameter supplied. "
00468             "Oscillator selection is unchanged."
00469         )
00470         logging.error(err)
00471         return err
00472     self.caca.setSubregister("OSC_SELECT", payload)
00473
00474 def setTriggerDelay(self, delay=0):
00475     """
00476     Use trigger delay timer. Actual delay is rounded down to multiple of .15 ns, up
00477     to a maximum delay of 6 ns
00478
00479     Args:
00480         delay: trigger delay in ns
00481
00482     Returns:
00483         String of errors, if any
00484     """
00485     logging.debug(self.logdebug + "setTriggerDelay; delay = " + str(delay))
00486     if (
00487         not (isinstance(delay, int) or isinstance(delay, float))
00488         or delay < 0
00489         or delay > 6
00490     ):
00491         err = (
00492             self.logerr + "invalid trigger delay submitted. Delay remains "
00493             "unchanged. "
00494         )
00495         logging.error(err)
00496         return err
00497     delayblocks = int(delay / 0.15)
00498     if delayblocks < 0:
00499         delayblocks = 0
00500     if delayblocks > 40:
00501         delayblocks = 40
00502     delayseq = (40 - delayblocks) * [0] + delayblocks * [1]
00503     seqstr = "".join(str(x) for x in delayseq)
00504     seqhex = "%x" % int(seqstr, 2)
00505     logging.debug(self.logdebug + "seqhex = " + str(seqhex))
00506     highpart = seqhex[-10:-8].zfill(8)
00507     lowpart = seqhex[-8:].zfill(8)
00508     self.caca.setSubregister("HST_DEL_EN", "1")
00509     err0, _ = self.caca.setRegister("HST_TRIGGER_DELAY_DATA_LO", lowpart)
00510     err1, _ = self.caca.setRegister("HST_TRIGGER_DELAY_DATA_HI", highpart)
00511     err2, _ = self.caca.setSubregister("HST_MODE", "1")
00512     delayed = delayblocks * 0.15
00513     logging.info(self.loginfo + "Actual trigger delay = " + str(delayed) + " ns")
00514     return err0 + err1 + err2
00515
00516 def setPhiDelay(self, side=None, delay=0):
00517     """
00518     Use phi delay timer. Actual delay is rounded down to multiple of .15 ns, up to a
00519     maximum delay of 1.5 ns
00520
00521     Args:
00522         side: hemisphere to delay; if None, delay both hemispheres
00523         delay: phi delay in ns
00524
00525     Returns:
00526         String of errors, if any
00527     """
00528     logging.debug(self.logdebug + "setPhiDelay; delay = " + str(delay))
00529     if (
00530         not (isinstance(delay, int) or isinstance(delay, float))
00531         or delay < 0
00532         or delay > 1.5
00533     ):
00534         err = (
00535             self.logerr + "invalid phi delay submitted. Delay remains "
00536             "unchanged. "
00537         )
00538         logging.error(err)
00539         return err

```

```

00539         delayblocks = int(delay / 0.15)
00540         if delayblocks < 0:
00541             delayblocks = 0
00542         if delayblocks > 10:
00543             delayblocks = 10
00544         delayseq = (10 - delayblocks) * [0] + delayblocks * [1]
00545         seqstr = "".join(str(x) for x in delayseq)
00546         err1 = ""
00547         err2 = ""
00548         if side is None or side.upper() == "A":
00549             err1, _ = self.caca.setSubregister("PHI_DELAY_A", seqstr)
00550         if side is None or side.upper() == "B":
00551             err2, _ = self.caca.setSubregister("PHI_DELAY_B", seqstr)
00552         delayed = delayblocks * 0.15
00553         logging.info(self.loginfo + "Actual phi delay = " + str(delayed) + " ns")
00554         return err1 + err2
00555
00556 def setExtClk(self, dilation=None, frequency=None):
00557     """
00558     Override the standard board clock with the external clock.
00559     Args:
00560         dilation: ratio of base frequency (500 MHz) to desired external clock
00561         frequency: Default is 25. Overridden if frequency parameter is provided
00562         frequency: Desired frequency for phi clock.
00563     Returns:
00564         error message as string
00565     """
00566     logging.debug(
00567         self.logdebug
00568         + "setExtClk; dilation = "
00569         + str(dilation)
00570         + "; frequency = "
00571         + str(frequency)
00572     )
00573     if not (isinstance(frequency, int) or isinstance(frequency, float)):
00574         err = (
00575             self.logerr
00576             + "invalid external clock frequency submitted. Clock is not "
00577             + "operating"
00578         )
00579         logging.error(err)
00580         return err
00581     self.caca.selectOscillator("external")
00582     if not dilation:
00583         dilation = 25
00584     if not frequency:
00585         frequency = 5e7 / float(dilation)
00586     count = 2e7 / float(frequency) - 1 # base phi clock is 20 MHz?
00587     if count < 0:
00588         count = 0
00589         warn = (
00590             self.logwarn
00591             + "external clock frequency exceeding maximum. Frequency set to "
00592             + "maximum (20 MHz)"
00593         )
00594         logging.warning(warn)
00595     if count > 0xFFFFFFFF:
00596         count = 0xFFFFFFFF
00597     counthex = hex(int(count))[2:].zfill(8)
00598     self.caca.setRegister("HST_EXT_CLK_HALF_PER", counthex)
00599
00600 # TODO: enable exponential form for all large number inputs (accept floats)
00601
00602 def setManualShutters(self, timing=None):
00603     """
00604     Legacy alias for setManualTiming()
00605     """
00606     self.setManualTimingsetManualTiming(timing)
00607
00608 def setManualTiming(self, timing=None):
00609     """
00610     Manual shutter timing, five intervals given in nanoseconds, e.g.,
00611     [100,50,100,50,100] for frame 0 open for 100 ns, an interframe pause of 50 ns,
00612     frame 1 open for 100 ns, etc. Timing is set for both hemispheres.
00613
00614     The actual timing is rounded down to the nearest multiple of 25 ns. (Each
00615     count = 25 ns. e.g., a request for 140 ns rounds down to a count of '5',
00616     which corresponds to 125 ns))
00617     - Minimum timing is 75 ns
00618     - Maximum is 25 * 2^30 ns (approximately 27 seconds)
00619

```



```

00620     Args:
00621         timing: 5-element list in nanoseconds
00622
00623     Returns:
00624         tuple (error string, response string from final message)
00625     """
00626     if timing is None:
00627         logging.info(
00628             self.logininfo
00629             + "No manual timing setting provided, defaulting to (100, 150, 100, "
00630             + " 150, 100, 150, 100) for both hemispheres"
00631         )
00632         timing = [(100, 150, 100, 150, 100)]
00633
00634     logging.info(self.logininfo + "Manual shutter sequence: " + str(timing))
00635     flattened = flattenList(timing)
00636     if (
00637         len(flattened) != 5
00638         or not all(isinstance(x, (int, float)) for x in flattened)
00639         or not all(x >= 25 for x in flattened)
00640     ):
00641         err = self.logerr + "Invalid manual shutter timing list: " + str(timing)
00642         logging.error(err + "; timing settings unchanged")
00643         return err, "00000000"
00644
00645     timecounts = [int(a // 25) for a in flattened]
00646     self.caca.sensmanual = timing
00647     self.caca.sens timing = {} # clear HST settings from ca object
00648
00649     control_messages = [
00650         ("MANSHUT_MODE", "1"),
00651         ("EXT_PHI_CLK_SH0_ON", "{0:#0{1}x}".format(timecounts[0], 10)[2:10]),
00652         ("EXT_PHI_CLK_SH0_OFF", "{0:#0{1}x}".format(timecounts[1], 10)[2:10]),
00653         ("EXT_PHI_CLK_SH1_ON", "{0:#0{1}x}".format(timecounts[2], 10)[2:10]),
00654         ("EXT_PHI_CLK_SH1_OFF", "{0:#0{1}x}".format(timecounts[3], 10)[2:10]),
00655         ("EXT_PHI_CLK_SH2_ON", "{0:#0{1}x}".format(timecounts[4], 10)[2:10]),
00656     ]
00657     return self.caca.submitMessages(control_messages, " setManualShutters: ")
00658
00659 def getManualTiming(self):
00660     """
00661     Read off manual shutter timing settings
00662     Returns:
00663         list of manual timing intervals
00664     """
00665     timing = []
00666     for reg in [
00667         "EXT_PHI_CLK_SH0_ON",
00668         "EXT_PHI_CLK_SH0_OFF",
00669         "EXT_PHI_CLK_SH1_ON",
00670         "EXT_PHI_CLK_SH1_OFF",
00671         "EXT_PHI_CLK_SH2_ON",
00672     ]:
00673         _, reghex = self.caca.getRegister(reg)
00674         timing.append(25 * int(reghex, 16))
00675     return timing
00676
00677 def getSensTemp(self, scale=None, offset=None, slope=None, dec=1):
00678     """
00679     Read temperature sensor located on the Daedalus sensor
00680     Args:
00681         scale: temperature scale to report (defaults to C, options are F and K)
00682         offset: offset of linear fit of sensor response (defaults to self.toffset)
00683         slope: slope of linear fit of sensor response (defaults to self.tslope)
00684         dec: round to 'dec' digits after the decimal point
00685
00686     Returns:
00687         temperature as float on given scale, rounded to .1 degree
00688     """
00689     err, rval = self.caca.getMonV("MON_TSENSE_OUT", errflag=True)
00690     if err:
00691         logging.error(
00692             self.logerr + "unable to retrieve temperature information ("
00693             + 'getTemp), returning "0" '
00694         )
00695         return 0.0
00696     if offset is None:
00697         offset = self.toffset
00698     if slope is None:
00699         slope = self.tslope
00700

```

```

00701         ctemp = offset + slope * rval
00702         if scale == "K":
00703             temp = round(ctemp + 273.15, dec)
00704         elif scale == "F":
00705             temp = round(1.8 * ctemp + 32, dec)
00706         else:
00707             temp = round(ctemp, dec)
00708         return temp
00709
00710     def parseReadoff(self, frames, columns):
00711         """
00712         Parses frames from board into images
00713         Args:
00714             frames: list of data arrays (frames) returned from board
00715             columns: 1 (full width image) or 2 (hemispheres generate distinct images)
00716         Returns:
00717             list of data arrays (frames) reordered and deinterlaced
00718         """
00719         logging.debug(self.logdebug + "parseReadoff")
00720         w = self.width
00721         if hasattr(self, "ca"): # TODO: this may no longer be necessary
00722             padIt = self.caca.padToFull
00723         else:
00724             padIt = self.padToFull
00725         if padIt:
00726             rows = self.maxheight
00727         else:
00728             rows = self.lastrow - self.firstrow + 1
00729         parsed = []
00730         for frame in frames:
00731             current = np.zeros((rows, w), dtype=np.uint16)
00732             mapped = np.zeros((rows, w), dtype=np.uint16)
00733             frame = frame.reshape(rows, w)
00734
00735             for entry in range(int(w / 2)):
00736                 col = 32 * (entry % 8) + entry // 8 # lookup from daedlookup.xls
00737                 for row in range(rows):
00738                     current[row][col] = frame[row][2 * entry]
00739                     current[row][col + 256] = frame[row][2 * entry + 1]
00740
00741             for row in range(rows):
00742                 mapped[row][0:32] = current[row][320:352]
00743                 mapped[row][32:64] = current[row][352:384]
00744                 mapped[row][64:96] = current[row][192:224]
00745                 mapped[row][96:128] = current[row][160:192]
00746                 mapped[row][128:160] = current[row][256:288]
00747                 mapped[row][160:192] = current[row][288:320]
00748                 mapped[row][192:224] = current[row][416:448]
00749                 mapped[row][224:256] = current[row][32:64]
00750                 mapped[row][256:288] = current[row][128:160]
00751                 mapped[row][288:320] = current[row][224:256]
00752                 mapped[row][320:352] = current[row][384:416]
00753                 mapped[row][352:384] = current[row][448:480]
00754                 mapped[row][384:416] = current[row][480:512]
00755                 mapped[row][416:448] = current[row][0:32]
00756                 mapped[row][448:480] = current[row][64:96]
00757                 mapped[row][480:512] = current[row][96:128]
00758             parsed.append(mapped)
00759
00760         images = self.caca.partition(parsed, columns)
00761         flatimages = [flattenlist(x) for x in images]
00762         return flatimages
00763
00764     def reportStatusSensor(self, statusbits, statusbits2):
00765         """
00766         Print status messages from sensor-specific bits of status register or object
00767         status flags
00768
00769         Args:
00770             statusbits: result of checkStatus()
00771             statusbits2: result of checkStatus2()
00772         """
00773         if int(statusbits[3]):
00774             print(self.loginfo + "SH0_rise_B_edge detected")
00775         if int(statusbits[4]):
00776             print(self.loginfo + "SH0_fall_B_edge detected")
00777         if int(statusbits[12]):
00778             print(self.loginfo + "RSLNALLWENB detected")
00779         if int(statusbits[15]):
00780             print(self.loginfo + "RSLNALLWENA detected")
00781         if self.HFWHFW:

```

```

00782         print(self.logininfo + "High Full Well mode active")
00783         # TODO: handle two hemispheres for ZDT
00784         elif self.ZDTZDT:
00785             print(self.logininfo + "Zero Dead Time mode active")
00786         elif self.interlacinginterlacing != [0, 0]:
00787             print(
00788                 "{logininfo}Interlacing active: {interlacing}".format(
00789                     logininfo=self.logininfo, interlacing=str(self.interlacinginterlacing)
00790                 )
00791             )
00792         if self.caca.sensmanual == []:
00793             print(
00794                 "{logininfo}High-speed timing: A:{Atiming}, B:{Btiming}".format(
00795                     logininfo=self.logininfo,
00796                     Atiming=self.getTiming(side="A", actual=True),
00797                     Btiming=self.getTiming(side="B", actual=True),
00798                 )
00799             )
00800         else:
00801             print(
00802                 "{logininfo}Manual timing set to {timing}".format(
00803                     logininfo=self.logininfo, timing=self.getManualTiminggetManualTiming()
00804                 )
00805             )
00806
00807
00808 """
00809 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00810 LLNL-CODE-838080
00811
00812 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00813 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00814 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00815 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00816 be made under this license.
00817 """

```

7.35 C:/Users/hill35/git/camera_python/nsCamera/sensors/icarus.py File Reference

Classes

- class `nsCamera.sensors.icarus.icarus`

Namespaces

- namespace `nsCamera`
- namespace `nsCamera.sensors`
- namespace `nsCamera.sensors.icarus`

7.36 icarus.py

[Go to the documentation of this file.](#)

```

00001 # -*- coding: utf-8 -*-
00002 """
00003 Parameters and functions specific to the icarus two-frame sensor
00004
00005 ***Do not use this file as a template for new code development***
00006
00007 Author: Jeremy Martin Hill (jerhill@llnl.gov)
00008
00009 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.

```

```

00010 LLNL-CODE-838080
00011
00012 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00013 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00014 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00015 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00016 be made under this license.
00017
00018 Version: 2.1.2 (February 2025)
00019 ""
00020
00021 import logging
00022 from collections import OrderedDict
00023
00024 from nsCamera.sensors.sensorBase import sensorBase
00025
00026
00027 class icarus(sensorBase):
00028     specwarn = " and the use of the Icarus model 1 sensor"
00029     minframe = 1 # fixed value for sensor
00030     maxframe = 2 # fixed value for sensor
00031     # WARNING: the camera will always 'acquire' four frames, but will only generate
00032     # images for the middle two; HST and manual shutters will manage all four
00033     # frames
00034     maxwidth = 512 # fixed value for sensor
00035     maxheight = 1024 # fixed value for sensor
00036     bytesperpixel = 2
00037     icarustype = 1 # 2-frame version
00038     fpганumID = 1 # last nybble of FPGA_NUM
00039     detect = "ICARUS_DET"
00040     sensfam = "Icarus"
00041     loglabel = "[Icarus1] "
00042     firstframe = 1
00043     lastframe = 2
00044     nframes = 2
00045     width = 512
00046     height = 1024
00047     firstrow = 0
00048     lastrow = 1023
00049     interlacing = [0, 0] # N/A for icarus
00050     columns = 1
00051     padToFull = True
00052
00053     def __init__(self, ca):
00054         self.caca = ca
00055         super(icarus, self).__init__(ca)
00056
00057         self.sens_registers = OrderedDict(
00058             {
00059                 "VRESET_WAIT_TIME": "03E",
00060                 "ICARUS_VER_SEL": "041",
00061                 "VRESET_HIGH_VALUE": "04A",
00062                 "MISC_SENSOR_CTL": "04C",
00063                 "MANUAL_SHUTTERS_MODE": "050",
00064                 "W0_INTEGRATION": "051",
00065                 "W0_INTERFRAME": "052",
00066                 "W1_INTEGRATION": "053",
00067                 "W1_INTERFRAME": "054",
00068                 "W2_INTEGRATION": "055",
00069                 "W2_INTERFRAME": "056",
00070                 "W3_INTEGRATION": "057",
00071                 "W0_INTEGRATION_B": "058",
00072                 "W0_INTERFRAME_B": "059",
00073                 "W1_INTEGRATION_B": "05A",
00074                 "W1_INTERFRAME_B": "05B",
00075                 "W2_INTEGRATION_B": "05C",
00076                 "W2_INTERFRAME_B": "05D",
00077                 "W3_INTEGRATION_B": "05E",
00078                 "TIME_ROW_DCD": "05F",
00079             }
00080         )
00081
00082         self.sens_subregisters = [
00083             ("MANSHUT_MODE", "MANUAL_SHUTTERS_MODE", 0, 1, True),
00084             ("REVREAD", "CTRL_REG", 4, 1, True),
00085             ("PDBIAS_LOW", "CTRL_REG", 6, 1, True),
00086             ("ROWDCD_CTL", "CTRL_REG", 7, 1, True),
00087             ("ACCUMULATION_CTL", "MISC_SENSOR_CTL", 0, 1, True),
00088             ("HST_TST_ANRST_EN", "MISC_SENSOR_CTL", 1, 1, True),
00089             ("HST_TST_BNRST_EN", "MISC_SENSOR_CTL", 2, 1, True),

```

```

00092         ("HST_TST_ANRST_IN", "MISC_SENSOR_CTL", 3, 1, True),
00093         ("HST_TST_BNRST_IN", "MISC_SENSOR_CTL", 4, 1, True),
00094         ("HST_PXL_RST_EN", "MISC_SENSOR_CTL", 5, 1, True),
00095         ("HST_CONT_MODE", "MISC_SENSOR_CTL", 6, 1, True),
00096         ("COL_DCD_EN", "MISC_SENSOR_CTL", 7, 1, True),
00097         ("COL_READOUT_EN", "MISC_SENSOR_CTL", 8, 1, True),
00098         ("READOFF_DELAY_EN", "TRIGGER_CTL", 4, 1, True),
00099
00102         ("STAT_W3TOPAEDGE1", "STAT_REG", 3, 1, False),
00103         ("STAT_W3TOPBEDGE1", "STAT_REG", 4, 1, False),
00104         ("STAT_HST_ALL_W_EN_DETECTED", "STAT_REG", 12, 1, False),
00105         ("PDBIAS_UNREADY", "STAT_REG2", 5, 1, False),
00106     ]
00107
00108     if self.caca.boardname == "llnl_v1":
00109         self.sens_subregisters.append(
00110             ("VRESET_HIGH", "VRESET_HIGH_VALUE", 7, 8, True)
00111         )
00112     else:
00113         self.sens_subregisters.extend(
00114             [
00115                 ("VRESET_HIGH", "VRESET_HIGH_VALUE", 15, 16, True),
00116                 ("READOFF_DELAY_EN", "TRIGGER_CTL", 4, 1, True),
00117             ]
00118         )
00119         self.sens_registers.update({"DELAY_ASSERTION_ROWDCD_EN": "04F"})
00120
00121     def checkSensorVoltStat(self):
00122         """
00123         Checks register tied to sensor select jumpers to confirm match with sensor
00124         object
00125
00126         Returns:
00127             boolean, True if jumpers select for Icarus sensor
00128         """
00129         logging.debug(self.logdebug + "checkSensorVoltStat")
00130         err, status = self.caca.getSubregister("ICARUS_DET")
00131         if err:
00132             logging.error(self.logerr + "unable to confirm sensor status")
00133             return False
00134         if not int(status):
00135             logging.error(self.logerr + "Icarus sensor not detected")
00136             return False
00137         return True
00138
00139     def sensorSpecific(self):
00140         """
00141         Returns:
00142             list of tuples, (Sensor-specific register, default setting)
00143         """
00144         icarussettings = [
00145             ("ICARUS_VER_SEL", "00000001"),
00146             ("FPA_FRAME_INITIAL", "00000001"),
00147             ("FPA_FRAME_FINAL", "00000002"),
00148             ("FPA_ROW_INITIAL", "00000000"),
00149             ("FPA_ROW_FINAL", "000003FF"),
00150             ("VRESET_WAIT_TIME", "000927C0"),
00151             ("HS_TIMING_DATA_BHI", "00000000"),
00152             ("HS_TIMING_DATA_BLO", "00006666"), # 0db6 = 2-1; 6666 = 2-2
00153             ("HS_TIMING_DATA_AHI", "00000000"),
00154             ("HS_TIMING_DATA_ALO", "00006666"),
00155         ]
00156         if self.caca.boardname == "llnl_v1":
00157             icarussettings.append(
00158                 ("VRESET_HIGH_VALUE", "000000D5") # 3.3 V (FF = 3.96)
00159             )
00160         else:
00161             icarussettings.append(("VRESET_HIGH_VALUE", "0000FFFF"))
00162         return icarussettings
00163
00164
00165     """
00166     Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00167     LLNL-CODE-838080
00168
00169     This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00170     contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00171     and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00172     'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00173     be made under this license.
00174     """

```

7.37 C:/Users/hill35/git/camera_python/nsCamera/sensors/icarus2.py File Reference

Classes

- class `nsCamera.sensors.icarus2.icarus2`

Namespaces

- namespace `nsCamera`
- namespace `nsCamera.sensors`
- namespace `nsCamera.sensors.icarus2`

7.38 icarus2.py

[Go to the documentation of this file.](#)

```
00001 # -*- coding: utf-8 -*-
00002 """
00003 Parameters and functions specific to the four-frame icarus2 sensor
00004
00005 Author: Jeremy Martin Hill (jerhill@llnl.gov)
00006
00007 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00008 LLNL-CODE-838080
00009
00010 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00011 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00012 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00013 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00014 be made under this license.
00015
00016 Version: 2.1.2 (February 2025)
00017 """
00018
00019 from collections import OrderedDict
00020
00021 from nsCamera.sensors.sensorBase import sensorBase
00022
00023
00024 class icarus2(sensorBase):
00025     specwarn = ""
00026     minframe = 0 # fixed value for sensor
00027     maxframe = 3 # fixed value for sensor
00028     maxwidth = 512 # fixed value for sensor
00029     maxheight = 1024 # fixed value for sensor
00030     bytesperpixel = 2
00031     icarustype = 0 # 4-frame version
00032     fpganumID = 1 # last nybble of FPGA_NUM
00033     detect = "ICARUS_DET"
00034     sensfam = "Icarus"
00035     loglabel = "[Icarus2] "
00036     firstframe = 0
00037     lastframe = 3
00038     nframes = 4
00039     width = 512
00040     height = 1024
00041     firstrow = 0
00042     lastrow = 1023
00043     interlacing = [0, 0] # N/A for icarus
00044     columns = 1
00045     padToFull = True
00046
00047     def __init__(self, ca):
00048         self.caca = ca
00049         super(icarus2, self).__init__(ca)
```

```

00050
00051     self.sens_registers = OrderedDict(
00052         {
00053             "VRESET_WAIT_TIME": "03E",
00054             "ICARUS_VER_SEL": "041",
00055             "MISC_SENSOR_CTL": "04C",
00056             "MANUAL_SHUTTERS_MODE": "050",
00057             "W0_INTEGRATION": "051",
00058             "W0_INTERFRAME": "052",
00059             "W1_INTEGRATION": "053",
00060             "W1_INTERFRAME": "054",
00061             "W2_INTEGRATION": "055",
00062             "W2_INTERFRAME": "056",
00063             "W3_INTEGRATION": "057",
00064             "W0_INTEGRATION_B": "058",
00065             "W0_INTERFRAME_B": "059",
00066             "W1_INTEGRATION_B": "05A",
00067             "W1_INTERFRAME_B": "05B",
00068             "W2_INTEGRATION_B": "05C",
00069             "W2_INTERFRAME_B": "05D",
00070             "W3_INTEGRATION_B": "05E",
00071             "TIME_ROW_DCD": "05F",
00072         }
00073     )
00074
00075     self.sens_subregisters = [
00076         ("MANSHUT_MODE", "MANUAL_SHUTTERS_MODE", 0, 1, True),
00077         ("REVREAD", "CTRL_REG", 4, 1, True),
00078         ("PDBIAS_LOW", "CTRL_REG", 6, 1, True),
00079         ("ROWDCD_CTL", "CTRL_REG", 7, 1, True),
00080         ("ACCUMULATION_CTL", "MISC_SENSOR_CTL", 0, 1, True),
00081         ("HST_TST_ANRST_EN", "MISC_SENSOR_CTL", 1, 1, True),
00082         ("HST_TST_BNRST_EN", "MISC_SENSOR_CTL", 2, 1, True),
00083         ("HST_TST_ANRST_IN", "MISC_SENSOR_CTL", 3, 1, True),
00084         ("HST_TST_BNRST_IN", "MISC_SENSOR_CTL", 4, 1, True),
00085         ("HST_PXL_RST_EN", "MISC_SENSOR_CTL", 5, 1, True),
00086         ("HST_CONT_MODE", "MISC_SENSOR_CTL", 6, 1, True),
00087         ("COL_DCD_EN", "MISC_SENSOR_CTL", 7, 1, True),
00088         ("COL_READOUT_EN", "MISC_SENSOR_CTL", 8, 1, True),
00089
00090         ("STAT_W3TOPAEDGE1", "STAT_REG", 3, 1, False),
00091         ("STAT_W3TOPBEDGE1", "STAT_REG", 4, 1, False),
00092         ("STAT_HST_ALL_W_EN_DETECTED", "STAT_REG", 12, 1, False),
00093         ("PDBIAS_UNREADY", "STAT_REG2", 5, 1, False),
00094     ]
00095
00096     if self.caca.boardname == "llnl_v4":
00097         self.sens_subregisters.append(
00098             ("READOFF_DELAY_EN", "TRIGGER_CTL", 4, 1, True)
00099         )
00100         self.sens_registers.update({"DELAY_ASSERTION_ROWDCD_EN": "04F"})
00101
00102     # TODO: clean up static methods
00103     def sensorSpecific(self):
00104         """
00105         Returns:
00106             list of tuples, (Sensor-specific register, default setting)
00107         """
00108         return [
00109             ("ICARUS_VER_SEL", "00000000"),
00110             ("FPA_FRAME_INITIAL", "00000000"),
00111             ("FPA_FRAME_FINAL", "00000003"),
00112             ("FPA_ROW_INITIAL", "00000000"),
00113             ("FPA_ROW_FINAL", "0000003FF"),
00114             ("HS_TIMING_DATA_BHI", "00000000"),
00115             ("HS_TIMING_DATA_BLO", "00006666"), # 0db6 = 2-1; 6666 = 2-2
00116             ("HS_TIMING_DATA_AHI", "00000000"),
00117             ("HS_TIMING_DATA_ALO", "00006666"),
00118         ]
00119
00120     """
00121     Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00122     LLNL-CODE-838080
00123
00124     This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00125     contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00126     and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00127     'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00128     be made under this license.

```

```
00134 """
```

7.39 C:/Users/hill35/git/camera_python/nsCamera/sensors/sensorBase.py File Reference

Classes

- class [nsCamera.sensors.sensorBase.sensorBase](#)

Namespaces

- namespace [nsCamera](#)
- namespace [nsCamera.sensors](#)
- namespace [nsCamera.sensors.sensorBase](#)

7.40 sensorBase.py

[Go to the documentation of this file.](#)

```
00001 # -*- coding: utf-8 -*-
00002 """
00003 Superclass for nsCamera sensors
00004
00005 Author: Jeremy Martin Hill (jerhill@llnl.gov)
00006
00007 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00008 LLNL-CODE-838080
00009
00010 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00011 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00012 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00013 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00014 be made under this license.
00015
00016 Version: 2.1.2 (February 2025)
00017 """
00018 import itertools
00019 import logging
00020 import numbers
00021
00022 from nsCamera.utils.misc import flattenlist, makeLogLabels
00023
00024
00025 class sensorBase(object):
00026     """
00027     Base class for sensors. 'Virtual' methods below default to Icarus behavior.
00028     daedalus.py overrides these methods as necessary
00029     """
00030
00031     def __init__(self, camassem):
00032         self.ca = camassem
00033         # skip board settings if no board object exists
00034         if hasattr(self.ca, "board"):
00035             self.init_board_specific()
00036
00037         (
00038             self.logcrit,
00039             self.logerr,
00040             self.logwarn,
00041             self.loginfo,
00042             self.logdebug,
00043         ) = makeLogLabels(self.ca.logtag, self.loglabel)
00044
```



```

00045         # skip assignment if no comms object exists
00046         if hasattr(self.ca, "comms"):
00047             self.ca.comms.payloadsize = (
00048                 self.width * self.height * self.nframes * self.bytesperpixel
00049             )
00050
00051         logging.info(self.loginfo + "Initializing sensor object")
00052
00053     def init_board_specific(self):
00054         """Initialize aliases and subregisters specific to the current board and sensor."""
00055
00056         if self.ca.sensorname == "icarus" or self.ca.sensorname == "icarus2":
00057             self.ca.board.subreg_aliases = self.ca.board.icarus_subreg_aliases
00058             self.ca.board.monitor_controls = self.ca.board.icarus_monitor_controls
00059         else:
00060             self.ca.board.subreg_aliases = self.ca.board.daedalus_subreg_aliases
00061             self.ca.board.monitor_controls = self.ca.board.daedalus_monitor_controls
00062
00063     # TODO: Check if 'jumpers' still apply for newer boards
00064     def checkSensorVoltStat(self):
00065         """
00066         Checks register tied to sensor select jumpers to confirm match with sensor
00067         object
00068
00069         Returns:
00070             boolean, True if jumpers select for Icarus sensor
00071         """
00072         logging.debug(self.logdebug + "checkSensorVoltStat")
00073         err, status = self.ca.getSubregister(self.detect)
00074         if err:
00075             logging.error(self.logerr + "Unable to confirm sensor status")
00076             return False
00077         if not int(status):
00078             logging.error(self.logerr + self.sensfam + " sensor not detected")
00079             return False
00080         return True
00081
00082     def setInterlacing(self, ifactor):
00083         """
00084         Virtual function; feature is not implemented on Icarus
00085         Overridden in daedalus.py
00086
00087         Returns:
00088             integer 0
00089         """
00090         if ifactor:
00091             logging.warning(
00092                 self.logwarn + "Interlacing is not supported by Icarus sensors. "
00093             )
00094         return 0
00095
00096     def setHighFullWell(self, flag):
00097         """
00098         Virtual function; feature is not implemented on Icarus
00099         Overridden in daedalus.py
00100         """
00101         if flag:
00102             logging.warning(
00103                 self.logwarn + "HighFullWell mode is not supported by Icarus sensors. "
00104             )
00105
00106     def setZeroDeadTime(self, flag):
00107         """
00108         Virtual function; feature is not implemented on Icarus
00109         Overridden in daedalus.py
00110         """
00111         if flag:
00112             logging.warning(
00113                 self.logwarn + "ZeroDeadTime mode is not supported by Icarus sensors. "
00114             )
00115
00116     def setTriggerDelay(self, delay):
00117         """
00118         Virtual function; feature is not implemented on Icarus
00119         Overridden in daedalus.py
00120         """
00121         if delay:
00122             logging.warning(
00123                 self.logwarn + "Trigger Delay is not supported by Icarus sensors. "
00124             )
00125

```

```

00126     def setPhiDelay(self, delay):
00127         """
00128         Virtual function; feature is not implemented on Icarus
00129         Overridden in daedalus.py
00130         """
00131         if delay:
00132             logging.warning(
00133                 self.logwarn + "Phi Delay is not supported by Icarus sensors. "
00134             )
00135
00136     def setExtClk(self, delay):
00137         """
00138         Virtual function; feature is not implemented on Icarus
00139         Overridden in daedalus.py
00140         """
00141         if delay:
00142             logging.warning(
00143                 self.logwarn + "External Phi Clock is not supported by Icarus sensors. "
00144             )
00145
00146     # TODO: update docs to reflect all sensors
00147     # TODO: remove default timing?
00148     # TODO: double-check o+c>20 ns in doc block
00149     def setTiming(self, side="AB", sequence=None, delay=0):
00150         """
00151         Sets timing registers based on 'sequence.' Requesting (0,0) timing will clear the
00152         timing register.
00153         *WARNING* if the entire sequence does not fit into the 40-bit register space,
00154         then the actual timings generated may differ from those requested. If the
00155         timing sequence fits only once into the register space (i.e., for a single
00156         frame, open + closed > 20 ns), then the actual timing will be (n, 40-n),
00157         irrespective of the setting of second parameter, e.g. (35,1) will actually
00158         result in (35,5) timing.
00159         *NOTE* Icarus sensors generally cannot use 1 ns timing, so all values for these
00160         devices (besides the delay) should be at least 2 ns
00161
00162         Args:
00163             side: Hemisphere 'A', 'B', 'AB'
00164             sequence: two-element tuple of timing durations in ns, e.g., '(5,2)'
00165             delay: initial delay in ns (1 ns delay is acceptable)
00166
00167         Returns:
00168             tuple (error string, 10-character hexadecimal representation of timing
00169                 sequence)
00170         """
00171         logging.info(
00172             "{}setTiming; side={}, sequence={}, delay={}".format(
00173                 self.loginfo, side, sequence, delay
00174             )
00175         )
00176         if sequence is None:
00177             sequence = (3, 2)
00178         if delay is None:
00179             delay = 0
00180         logging.info(
00181             self.loginfo
00182             + "HST side "
00183             + side.upper()
00184             + ": "
00185             + str(sequence)
00186             + "; delay = "
00187             + str(delay)
00188         )
00189         err = ""
00190         if len(sequence) != 2:
00191             err = (
00192                 self.logerr
00193                 + "Invalid sequence setting for side: "
00194                 + side
00195                 + "; timing settings are unchanged"
00196             )
00197             logging.error(err)
00198             return err, "0000000000"
00199         if side.upper() == "AB":
00200             err1, _ = self.setTiming(side="A", sequence=sequence, delay=delay)
00201             err2, full40hex = self.setTiming(side="B", sequence=sequence, delay=delay)
00202             return err1 + err2, full40hex
00203         if side.upper() == "A":
00204             lowreg = "HS_TIMING_DATA_ALO"
00205             highreg = "HS_TIMING_DATA_AHI"
00206         elif side.upper() == "B":

```

```

00207         lowreg = "HS_TIMING_DATA_BLO"
00208         highreg = "HS_TIMING_DATA_BHI"
00209     else:
00210         err = (
00211             self.logerr
00212             + "setTiming: Invalid sensor side: "
00213             + side
00214             + "; timing settings unchanged"
00215         )
00216         logging.error(err)
00217         return err, "0000000000"
00218     if (sequence[0] + sequence[1]) + delay > 40:
00219         err = (
00220             self.logerr
00221             + "setTiming: Timing sequence is too long to be implemented; "
00222             + "timing settings unchanged "
00223         )
00224         logging.error(err)
00225         return err, "0000000000"
00226
00227     self.ca.senstiming[side.upper()] = (sequence, delay)
00228     self.ca.sensmanual = [] # clear manual settings from ca
00229
00230     full40 = [0] * 40
00231     bitlist = []
00232     flag = 1
00233     sequence = sequence[:2] # TODO: is this redundant?
00234     for a in sequence:
00235         add = [flag] * a
00236         bitlist += add
00237         if flag:
00238             flag = 0
00239         else:
00240             flag = 1
00241     # automatically truncates sequence to 39 characters
00242     logging.debug(self.logdebug + "bitlist = " + str(bitlist))
00243     if bitlist: # skip this if timing is [0,0]
00244         reversedlist = bitlist[39::-1]
00245         trunclist = reversedlist[:]
00246         while trunclist[0] == 0:
00247             trunclist.pop(0)
00248         # fullrepeat counts open/closed cycles, doesn't include final frame
00249         fullrepeats = (40 - len(trunclist) - delay) // len(reversedlist)
00250         logging.debug(self.logdebug + "fullrepeats = " + str(fullrepeats))
00251         # Pattern from sequence repeated to fit inside 40 bits
00252         repeated = trunclist + reversedlist * fullrepeats
00253         full40[-(len(repeated) + delay + 1) : -(delay + 1)] = repeated
00254     else:
00255         logging.warning(self.logwarn + "setTiming: all-zero timing supplied")
00256         fullrepeats = self.nframes
00257         full40bin = "".join(str(x) for x in full40)
00258         logging.debug(self.logdebug + "full40bin = " + str(full40bin))
00259         full40hex = "%x" % int(full40bin, 2)
00260         logging.debug(self.logdebug + "full40hex = " + str(full40hex))
00261         highpart = full40hex[-10:-8].zfill(8)
00262         lowpart = full40hex[-8:].zfill(8)
00263         err0, _ = self.ca.setRegister(lowreg, lowpart)
00264         err1, _ = self.ca.setRegister(highreg, highpart)
00265         err2, _ = self.ca.setSubregister("MANSHUT_MODE", "0")
00266         err3, _ = self.ca.setSubregister("HST_MODE", "1")
00267         err = err0 + err1 + err2 + err3
00268     if err:
00269         logging.error(
00270             self.logerr + "setTiming: Timing may not have been set correctly"
00271         )
00272     if fullrepeats < self.nframes - 1:
00273         actual = self.getTiming(side, actual=True)
00274         if self.fpganumID == 2:
00275             expected = [delay] + 2 * list(sequence) + [sequence[0]]
00276         else:
00277             expected = [delay] + 3 * list(sequence) + [sequence[0]]
00278         if actual != expected:
00279             logging.warning(
00280                 self.logwarn
00281                 + "setTiming: Due to sequence length"
00282                 + self.specwarn
00283                 + ", the actual timing "
00284                 + "sequence for side "
00285                 + side
00286                 + " will be "
00287                 + "{"

```

```

00288         + str(actual[0])
00289         + "}"
00290         + " "
00291         + str(actual[1 : 2 * self.nframes])
00292     )
00293     elif self.ca.sensorname == "Icarus":
00294         f0delay = sequence[0] + sequence[1]
00295         logging.warning(
00296             self.logwarn + "setTiming: Due to use of the Icarus model 1 sensor, the"
00297             " initial delay for side "
00298             + side
00299             + " will actually be "
00300             + str(delay + f0delay)
00301             + " nanoseconds"
00302         )
00303     return err, full40hex
00304
00305     # TODO: restore after power cycle?
00306     # TODO: smart interpretation of Icarus1 timing?
00307     # TODO: error checking like in getTiming
00308 def setArbTiming(self, side="AB", sequence=None):
00309     """
00310     Set arbitrary high-speed timing sequence.
00311     Args:
00312         side: Hemisphere 'A', 'B', 'AB'
00313         sequence: list of arbitrary timing intervals, beginning with initial delay.
00314             The conventional timing (3,2) with delay = 0 would be represented by
00315             [0,3,2,3,2,3,2,3] on icarus devices, [0,3,2,3,2,3] on daedalus. If used
00316             for interlacing or ZDT, you should populate the entire 40-bit register,
00317             e.g., [0,3,2,3,2,3,2,3,2,3,2,3,2,3,2,3,2,3,2,3,2,3,2,3,2,3,2,3,2,3,2,3,2,3,2,3]
00318
00319     *NOTE* Icarus sensors generally cannot use 1 ns timing, so should use at
00320     least 2 ns for frames 2 and 3 integration and interframe times (an initial
00321     delay of only 1 ns is acceptable)
00322
00323     *NOTE* although the Icarus model 1 only images the middle two frames, timing
00324     entries must be provided for all four frames; to implement frame 1 open
00325     for X ns, shutter closed for Y ns, and frame 2 open for Z ns, use the
00326     sequence [0,1,1,X,Y,Z,1,1]
00327
00328     *WARNING* arbitrary timings will not be restored after a board power cycle
00329
00330     Returns:
00331         list: Actual timing results
00332     """
00333     logging.info(
00334         "{}setArbTiming; side={}, sequence={}".format(self.loginfo, side, sequence)
00335     )
00336     if sequence is None:
00337         if self.sensfam == "Daedalus":
00338             sequence = [0, 2, 3, 4, 5, 6]
00339         else:
00340             sequence = [0, 2, 3, 4, 5, 6, 7, 8]
00341     logging.info(
00342         self.loginfo + "HST side " + side.upper() + " (arbitrary): " + str(sequence)
00343     )
00344     if side.upper() == "AB":
00345         err1, _ = self.setArbTiming(side="A", sequence=sequence)
00346         err2, actual = self.setArbTiming(side="B", sequence=sequence)
00347         return err1 + err2, actual
00348     if side.upper() == "A":
00349         lowreg = "HS_TIMING_DATA_ALO"
00350         highreg = "HS_TIMING_DATA_AHI"
00351     elif side.upper() == "B":
00352         lowreg = "HS_TIMING_DATA_BLO"
00353         highreg = "HS_TIMING_DATA_BHI"
00354     else:
00355         err = (
00356             self.logerr
00357             + "Invalid sensor side: "
00358             + side
00359             + "; timing settings unchanged"
00360         )
00361     logging.error("{}setArbTiming: {}".format(self.logerr, err))
00362     return err, "0000000000"
00363
00364     full40 = [0] * 40
00365     bitlist = []
00366     flag = 0 # similar to setTiming, but starts with delay
00367
00368     for a in sequence:

```

```

00369         add = [flag] * a
00370         bitlist += add
00371         if flag:
00372             flag = 0
00373         else:
00374             flag = 1
00375
00376         logging.debug(self.logdebug + "bitlist = " + str(bitlist))
00377         reversedlist = bitlist[39::-1]
00378         full40[-(len(reversedlist) + 1) : -1] = reversedlist
00379         full40bin = "".join(str(x) for x in full40)
00380         logging.debug(self.logdebug + "full40bin = " + str(full40bin))
00381         full40hex = "%x" % int(full40bin, 2)
00382         logging.debug(self.logdebug + "full40hex = " + str(full40hex))
00383         highpart = full40hex[-10:-8].zfill(8)
00384         lowpart = full40hex[-8:].zfill(8)
00385         self.ca.setRegister(lowreg, lowpart)
00386         self.ca.setRegister(highreg, highpart)
00387         # deactivates manual shutter mode if previously engaged
00388         self.ca.setSubregister("MANSHUT_MODE", "0")
00389         self.ca.setSubregister("HST_MODE", "1")
00390         actual = self.getTiming(side, actual=True)
00391         f0delay = sequence[1] + sequence[2]
00392
00393         if self.ca.sensorname == "icarus":
00394             if actual != sequence[:1] + sequence[3:6]:
00395                 logging.warning(
00396                     self.logwarn + "Due to sequence length and use of the Icarus model "
00397                     "1 sensor, the actual timing sequence for side "
00398                     + side
00399                     + " will be "
00400                     + "{"
00401                     + str(actual[0] + f0delay)
00402                     + "}"
00403                     + " "
00404                     + str(actual[1 : 2 * self.nframes])
00405                 )
00406             else:
00407                 logging.warning(
00408                     self.logwarn + "Due to use of the Icarus model 1 sensor, the actual"
00409                     " timing sequence for side "
00410                     + side
00411                     + " will be "
00412                     + "{"
00413                     + str(actual[0] + f0delay)
00414                     + "}"
00415                     + " "
00416                     + str(actual[1 : 2 * self.nframes])
00417                 )
00418         else:
00419             if actual != sequence:
00420                 logging.warning(
00421                     self.logwarn + "Due to sequence length, actual timing sequence "
00422                     "for side "
00423                     + side
00424                     + " will be "
00425                     + "{"
00426                     + str(actual[0])
00427                     + "}"
00428                     + " "
00429                     + str(actual[1 : 2 * self.nframes])
00430                 )
00431         return "", actual
00432
00433         # TODO: figure out how to handle interlacing?
00434     def getTiming(self, side, actual):
00435         """
00436         actual = True: returns actual high speed intervals that will be generated by the
00437                     FPGA as list
00438         False: Returns high speed timing settings as set by setTiming. Assumes
00439               that timing was set via the setTiming method--it will not accurately
00440               report arbitrary timings set by direct register sets or manual
00441               shutter control
00442
00443         Args:
00444             side: Hemisphere 'A' or 'B'
00445             actual: False: return HST settings
00446                   True: calculate and return actual HST behavior
00447
00448         Returns:
00449             actual= True: list of shutter intervals;

```

```

00450             icarus: [delay, open0, closed0, open1, closed1, open2, closed2,
00451                       open3]
00452             daedalus: [delay, open0, closed0, open1, closed1, open2]
00453             False: tuple (hemisphere label,
00454                           'open shutter' in ns,
00455                           'closed shutter' in ns,
00456                           initial delay in ns)
00457
00458     """
00459     logging.info("{}getTiming".format(self.logininfo))
00460     if side is None:
00461         side = "A"
00462     logging.info(self.logininfo + "get timing, side " + side.upper())
00463     if side.upper() == "A":
00464         lowreg = "HS_TIMING_DATA_ALO"
00465         highreg = "HS_TIMING_DATA_AHI"
00466     elif side.upper() == "B":
00467         lowreg = "HS_TIMING_DATA_BLO"
00468         highreg = "HS_TIMING_DATA_BHI"
00469     else:
00470         logging.error(
00471             self.logerr
00472             + "Invalid sensor side: "
00473             + side
00474             + "; timing settings unchanged"
00475         )
00476     return "", 0, 0, 0
00477     err, lowpart = self.ca.getRegister(lowreg)
00478     err1, highpart = self.ca.getRegister(highreg)
00479     if err or err1:
00480         logging.error(
00481             self.logerr + "Unable to retrieve timing setting (getTiming), "
00482             "returning zeroes "
00483         )
00484     return side.upper(), 0, 0, 0
00485     full40hex = highpart[-2:] + lowpart.zfill(8)
00486     full40bin = "{0:0=40b}".format(int(full40hex, 16))
00487     logging.debug(self.logdebug + "full40bin = " + str(full40bin))
00488     if actual:
00489         if full40bin == "0" * 40: # all-zero timing
00490             if self.fpganumID == 2:
00491                 times = [0] * 6
00492             else:
00493                 times = [0] * 8
00494         else:
00495             full160 = 4 * full40bin
00496             gblist = [[k, len(list(g))] for k, g in itertools.groupby(full160)]
00497             if self.fpganumID == 2:
00498                 times = [int(x[1]) for x in gblist[:-7:-1]]
00499             else:
00500                 times = [int(x[1]) for x in gblist[:-9:-1]]
00501             times[0] = times[0] - 1
00502         if self.ca.sensorname == "icarus":
00503             # get timing for frames 1 and 2, keep delay as offset
00504             # TODO: should this give a 'correct' offset from frame 0?
00505             times12 = [times[0]] + times[3:6]
00506             return times12
00507         return times
00508     else:
00509         if full40bin == "0" * 40: # all-zero timing
00510             timeon, timeoff, delay = (0, 0, 0)
00511         else:
00512             gblist = [[k, len(list(g))] for k, g in itertools.groupby(full40bin)]
00513             delay = gblist[-1][1] - 1
00514             timeon = gblist[-2][1]
00515
00516             if self.ca.sensorname == "icarus":
00517                 if len(gblist) == 2: # 39,1 corner case
00518                     timeoff = 1
00519                 elif len(gblist) == 3: # sequence fits only once
00520                     timeoff = 40 - timeon
00521                 else:
00522                     timeoff = gblist[-3][1]
00523             else:
00524                 if len(gblist) < self.nframes: # sequence fits only once
00525                     timeoff = 40 - timeon
00526                 else:
00527                     # TODO: confirm '-3' works for daedalus
00528                     timeoff = gblist[-3][1]
00529     return side.upper(), timeon, timeoff, delay
00530

```

```

00531     def setManualShutters(self, timing=None):
00532         """
00533         Legacy alias for setManualTiming()
00534         """
00535         self.setManualTiming(timing)
00536
00537     def setManualTiming(self, timing=None):
00538         """
00539         Manual shutter timing, seven intervals to assign to both hemispheres, e.g.,
00540         [(100,150,100,150,100,150,100)] for frame 0 open for 100 ns, an interframe
00541         pause of 50 ns, frame 1 open for 100 ns, etc.
00542         Provide two sets of seven intervals, e.g., [(100,150,100,150,100,150,100),
00543         (200,250,200,250,200,250,200)] to program the A and B hemispheres
00544         independently
00545
00546         Overridden in daedalus.py
00547
00548         The timing list is flattened before processing; the suggested tuple structure is
00549         just for clarity (first tuple is A, second is B) and is optional.
00550
00551         The actual timing is rounded down to the nearest multiple of 25 ns. (Each
00552         count = 25 ns. e.g., a request for 140 ns rounds down to a count of '5',
00553         which corresponds to 125 ns))
00554         - Minimum timing is 75 ns
00555         - Maximum is 25 * 2^30 ns (approximately 27 seconds)
00556
00557         Args:
00558             timing: 7- or 14-element list (substructure optional) in nanoseconds
00559
00560         Returns:
00561             tuple (error string, response string from final message)
00562         """
00563         if timing is None:
00564             logging.info(
00565                 self.loginf +
00566                 + "No manual timing setting provided, defaulting to (100, 150, 100, "
00567                 + "150, 100, 150, 100) for both hemispheres"
00568             )
00569             timing = [
00570                 (100, 150, 100, 150, 100, 150, 100),
00571                 (100, 150, 100, 150, 100, 150, 100),
00572             ]
00573             logging.info(self.loginf + "Manual shutter sequence: " + str(timing))
00574             flattened = flattenList(timing)
00575             if len(flattened) == 7:
00576                 flattened = 2 * flattened
00577             if (
00578                 len(flattened) != 14
00579                 or not all(isinstance(x, numbers.Real) for x in flattened)
00580                 or not all(x >= 75 for x in flattened)
00581                 or not all(x <= 26843545600 for x in flattened)
00582             ):
00583                 err = self.logerr + "Invalid manual shutter timing list: " + str(timing)
00584                 logging.error(err + "; timing settings unchanged")
00585                 return err, "00000000"
00586
00587             timecounts = [int(a // 25) for a in flattened]
00588             self.ca.sensmanual = timing
00589             self.ca.sens timing = {} # clear HST settings from ca object
00590
00591             control_messages = [
00592                 ("W0_INTEGRATION", "{0:#0{1}x}".format(timecounts[0], 10)[2:10]),
00593                 ("W0_INTERFRAME", "{0:#0{1}x}".format(timecounts[1], 10)[2:10]),
00594                 ("W1_INTEGRATION", "{0:#0{1}x}".format(timecounts[2], 10)[2:10]),
00595                 ("W1_INTERFRAME", "{0:#0{1}x}".format(timecounts[3], 10)[2:10]),
00596                 ("W2_INTEGRATION", "{0:#0{1}x}".format(timecounts[4], 10)[2:10]),
00597                 ("W2_INTERFRAME", "{0:#0{1}x}".format(timecounts[5], 10)[2:10]),
00598                 ("W3_INTEGRATION", "{0:#0{1}x}".format(timecounts[6], 10)[2:10]),
00599                 ("W0_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[7], 10)[2:10]),
00600                 ("W0_INTERFRAME_B", "{0:#0{1}x}".format(timecounts[8], 10)[2:10]),
00601                 ("W1_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[9], 10)[2:10]),
00602                 ("W1_INTERFRAME_B", "{0:#0{1}x}".format(timecounts[10], 10)[2:10]),
00603                 ("W2_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[11], 10)[2:10]),
00604                 ("W2_INTERFRAME_B", "{0:#0{1}x}".format(timecounts[12], 10)[2:10]),
00605                 ("W3_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[13], 10)[2:10]),
00606                 ("HST_MODE", "0"),
00607                 ("MANSHUT_MODE", "1"),
00608             ]
00609             return self.ca.submitMessages(control_messages, " setManualShutters: ")
00610
00611     def getManualTiming(self):

```

```

00612         """
00613         Read off manual shutter timing settings
00614         Overridden in daedalus.py
00615         Returns:
00616             list of 2 lists of timing from A and B sides, respectively
00617         """
00618         aside = []
00619         bside = []
00620         for reg in [
00621             "W0_INTEGRATION",
00622             "W0_INTERFRAME",
00623             "W1_INTEGRATION",
00624             "W1_INTERFRAME",
00625             "W2_INTEGRATION",
00626             "W2_INTERFRAME",
00627             "W3_INTEGRATION",
00628         ]:
00629             _, reghex = self.ca.getRegister(reg)
00630             aside.append(25 * int(reghex, 16))
00631         for reg in [
00632             "W0_INTEGRATION_B",
00633             "W0_INTERFRAME_B",
00634             "W1_INTEGRATION_B",
00635             "W1_INTERFRAME_B",
00636             "W2_INTEGRATION_B",
00637             "W2_INTERFRAME_B",
00638             "W3_INTEGRATION_B",
00639         ]:
00640             _, reghex = self.ca.getRegister(reg)
00641             bside.append(25 * int(reghex, 16))
00642         return [aside, bside]
00643
00644     def getSensTemp(self, scale=None, offset=None, slope=None, dec=None):
00645         """
00646         Virtual method (Temperature sensor is not present on Icarus sensors). Returns 0.
00647         Overridden by Daedalus method
00648         """
00649         return 0
00650
00651     def selectOscillator(self, osc=None):
00652         """
00653         Selects oscillator to control sensor timing
00654         Overridden in daedalus.py
00655         Args:
00656             osc: 'relaxation' | 'ring' | 'ringnoosc' | 'external', defaults to relaxation
00657
00658         Returns:
00659             error message as string
00660         """
00661         logging.info(self.loginfo + "selectOscillator; osc = " + str(osc))
00662         if osc is None:
00663             osc = "rel"
00664         osc = str(osc)
00665         if osc.upper()[3] == "REL":
00666             payload = "00"
00667         elif osc.upper()[3] == "RIN":
00668             if "NO" in osc.upper() or "0" in osc:
00669                 payload = "10"
00670             else:
00671                 payload = "01"
00672         elif osc.lower()[3] in ["ext", "phi"]:
00673             payload = "11"
00674         else:
00675             err = (
00676                 self.logerr + "selectOscillator: invalid parameter supplied. "
00677                 "Oscillator selection is unchanged."
00678             )
00679             logging.error(err)
00680             return err
00681         self.ca.setSubregister("OSC_SELECT", payload)
00682
00683     def parseReadoff(self, frames, columns):
00684         """
00685         Virtual method (Order parsing is unnecessary for Icarus, continue to hemisphere
00686         parsing.)
00687         Overridden by Daedalus method
00688         """
00689         return self.ca.partition(frames, columns)
00690
00691     def getSensorStatus(self):
00692         """

```



```

00693         Wrapper for reportSensorStatus so that the user doesn't have to query statusbits
00694         """
00695         sb1 = self.ca.board.checkstatus()
00696         sb2 = self.ca.board.checkstatus2()
00697         self.reportStatusSensor(sb1, sb2)
00698
00699     def reportStatusSensor(self, statusbits, statusbits2):
00700         """
00701         Print status messages from sensor-specific bits of status register, default for
00702         Icarus family sensors
00703         Args:
00704             statusbits: result of checkStatus()
00705             statusbits2: result of checkStatus2()
00706         """
00707         if int(statusbits[3]):
00708             print(self.loginfo + "W3_Top_A_Edge1 detected")
00709         if int(statusbits[4]):
00710             print(self.loginfo + "W3_Top_B_Edge1 detected")
00711         if int(statusbits[12]):
00712             print(self.loginfo + "HST_All_W_En detected")
00713         if self.ca.boardname == "llnl_v4" and int(statusbits2[5]):
00714             print(self.loginfo + "PDBIAS Unready")
00715
00716
00717     # TODO: add function to control TIME_ROW_DCD delay
00718
00719     """
00720     Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00721     LLNL-CODE-838080
00722
00723     This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00724     contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00725     and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00726     'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00727     be made under this license.
00728     """

```

7.41 C:/Users/hill35/git/camera_python/nsCamera/utils/crc16pure.py File Reference

Namespaces

- namespace [nsCamera](#)
- namespace [nsCamera.utils](#)
- namespace [nsCamera.utils.crc16pure](#)

Functions

- [nsCamera.utils.crc16pure._crc16](#) (data, crc, table)
- [nsCamera.utils.crc16pure.crc16xmodem](#) (data, crc=0)

Variables

- list [nsCamera.utils.crc16pure.CRC16_XMODEM_TABLE](#)

7.42 crc16pure.py

[Go to the documentation of this file.](#)

```
00001 #!/usr/bin/env python
00002 """Pure python library for calculating CRC16
00003     NOTE: modified slightly to combine Python 2 and Python 3 versions in single file
00004 """
00005
00006
00024
00025 import sys
00026
00027 # table for calculating CRC
00028 # this particular table was generated using pycrc v0.7.6, http://www.ttyl.net/pycrc/
00029 # using the configuration:
00030 # * Width = 16
00031 # * Poly = 0x1021
00032 # * XorIn = 0x0000
00033 # * ReflectIn = False
00034 # * XorOut = 0x0000
00035 # * ReflectOut = False
00036 # * Algorithm = table-driven
00037 # by following command:
00038 # python pycrc.py --model xmodem --algorithm table-driven --generate c
00039 CRC16_XMODEM_TABLE = [
00040     0x0000,
00041     0x1021,
00042     0x2042,
00043     0x3063,
00044     0x4084,
00045     0x50A5,
00046     0x60C6,
00047     0x70E7,
00048     0x8108,
00049     0x9129,
00050     0xA14A,
00051     0xB16B,
00052     0xC18C,
00053     0xD1AD,
00054     0xE1CE,
00055     0xF1EF,
00056     0x1231,
00057     0x0210,
00058     0x3273,
00059     0x2252,
00060     0x52B5,
00061     0x4294,
00062     0x72F7,
00063     0x62D6,
00064     0x9339,
00065     0x8318,
00066     0xB37B,
00067     0xA35A,
00068     0xD3BD,
00069     0xC39C,
00070     0xF3FF,
00071     0xE3DE,
00072     0x2462,
00073     0x3443,
00074     0x0420,
00075     0x1401,
00076     0x64E6,
00077     0x74C7,
00078     0x44A4,
00079     0x5485,
00080     0xA56A,
00081     0xB54B,
00082     0x8528,
00083     0x9509,
00084     0xE5EE,
00085     0xF5CF,
00086     0xC5AC,
00087     0xD58D,
00088     0x3653,
00089     0x2672,
00090     0x1611,
00091     0x0630,
00092     0x76D7,
00093     0x66F6,
```

```
00094      0x5695,
00095      0x46B4,
00096      0xB75B,
00097      0xA77A,
00098      0x9719,
00099      0x8738,
00100      0xF7DF,
00101      0xE7FE,
00102      0xD79D,
00103      0xC7BC,
00104      0x48C4,
00105      0x58E5,
00106      0x6886,
00107      0x78A7,
00108      0x0840,
00109      0x1861,
00110      0x2802,
00111      0x3823,
00112      0xC9CC,
00113      0xD9ED,
00114      0xE98E,
00115      0xF9AF,
00116      0x8948,
00117      0x9969,
00118      0xA90A,
00119      0xB92B,
00120      0x5AF5,
00121      0x4AD4,
00122      0x7AB7,
00123      0x6A96,
00124      0x1A71,
00125      0x0A50,
00126      0x3A33,
00127      0x2A12,
00128      0xDBFD,
00129      0xCBDC,
00130      0xFBBF,
00131      0xEB9E,
00132      0x9B79,
00133      0x8B58,
00134      0xBB3B,
00135      0xAB1A,
00136      0x6CA6,
00137      0x7C87,
00138      0x4CE4,
00139      0x5CC5,
00140      0x2C22,
00141      0x3C03,
00142      0x0C60,
00143      0x1C41,
00144      0xEDAE,
00145      0xFD8F,
00146      0xCDEC,
00147      0xDDCD,
00148      0xAD2A,
00149      0xBD0B,
00150      0x8D68,
00151      0x9D49,
00152      0x7E97,
00153      0x6EB6,
00154      0x5ED5,
00155      0x4EF4,
00156      0x3E13,
00157      0x2E32,
00158      0x1E51,
00159      0x0E70,
00160      0xFF9F,
00161      0xEFBE,
00162      0xDFDD,
00163      0xCFFC,
00164      0xBF1B,
00165      0xAF3A,
00166      0x9F59,
00167      0x8F78,
00168      0x9188,
00169      0x81A9,
00170      0xB1CA,
00171      0xA1EB,
00172      0xD10C,
00173      0xC12D,
00174      0xF14E,
```

00175	0xE16F,
00176	0x1080,
00177	0x00A1,
00178	0x30C2,
00179	0x20E3,
00180	0x5004,
00181	0x4025,
00182	0x7046,
00183	0x6067,
00184	0x83B9,
00185	0x9398,
00186	0xA3FB,
00187	0xB3DA,
00188	0xC33D,
00189	0xD31C,
00190	0xE37F,
00191	0xF35E,
00192	0x02B1,
00193	0x1290,
00194	0x22F3,
00195	0x32D2,
00196	0x4235,
00197	0x5214,
00198	0x6277,
00199	0x7256,
00200	0xB5EA,
00201	0xA5CB,
00202	0x95A8,
00203	0x8589,
00204	0xF56E,
00205	0xE54F,
00206	0xD52C,
00207	0xC50D,
00208	0x34E2,
00209	0x24C3,
00210	0x14A0,
00211	0x0481,
00212	0x7466,
00213	0x6447,
00214	0x5424,
00215	0x4405,
00216	0xA7DB,
00217	0xB7FA,
00218	0x8799,
00219	0x97B8,
00220	0xE75F,
00221	0xF77E,
00222	0xC71D,
00223	0xD73C,
00224	0x26D3,
00225	0x36F2,
00226	0x0691,
00227	0x16B0,
00228	0x6657,
00229	0x7676,
00230	0x4615,
00231	0x5634,
00232	0xD94C,
00233	0xC96D,
00234	0xF90E,
00235	0xE92F,
00236	0x99C8,
00237	0x89E9,
00238	0xB98A,
00239	0xA9AB,
00240	0x5844,
00241	0x4865,
00242	0x7806,
00243	0x6827,
00244	0x18C0,
00245	0x08E1,
00246	0x3882,
00247	0x28A3,
00248	0xCB7D,
00249	0xDB5C,
00250	0xEB3F,
00251	0xFB1E,
00252	0x8BF9,
00253	0x9BD8,
00254	0xABBB,
00255	0xBB9A,

```

00256     0x4A75,
00257     0x5A54,
00258     0x6A37,
00259     0x7A16,
00260     0x0AF1,
00261     0x1AD0,
00262     0x2AB3,
00263     0x3A92,
00264     0xFD2E,
00265     0xED0F,
00266     0xDD6C,
00267     0xCD4D,
00268     0xBDAA,
00269     0xAD8B,
00270     0x9DE8,
00271     0x8DC9,
00272     0x7C26,
00273     0x6C07,
00274     0x5C64,
00275     0x4C45,
00276     0x3CA2,
00277     0x2C83,
00278     0x1CE0,
00279     0x0CC1,
00280     0xEF1F,
00281     0xFF3E,
00282     0xCF5D,
00283     0xDF7C,
00284     0xAF9B,
00285     0xBFBA,
00286     0x8FD9,
00287     0x9FF8,
00288     0x6E17,
00289     0x7E36,
00290     0x4E55,
00291     0x5E74,
00292     0x2E93,
00293     0x3EB2,
00294     0x0ED1,
00295     0x1EF0,
00296 ]
00297
00298
00299 def _crc16(data, crc, table):
00300     """Calculate CRC16 using the given table.
00301     `data`         - data for calculating CRC, must be a string
00302     `crc`          - initial value
00303     `table`        - table for calculating CRC (list of 256 integers)
00304     Return calculated value of CRC
00305     """
00306     for byte in data:
00307         if sys.version_info > (3,):
00308             crc = ((crc << 8) & 0xFF00) ^ table[((crc >> 8) & 0xFF) ^ byte]
00309         else:
00310             crc = ((crc << 8) & 0xFF00) ^ table[((crc >> 8) & 0xFF) ^ ord(byte)]
00311
00312     return crc & 0xFFFF
00313
00314
00315 def crc16xmodem(data, crc=0):
00316     """Calculate CRC-CCITT (XModem) variant of CRC16.
00317     `data`         - data for calculating CRC, must be a string
00318     `crc`          - initial value
00319     Return calculated value of CRC
00320     """
00321     return _crc16(data, crc, CRC16_XMODEM_TABLE)

```

7.43 C:/Users/hill35/git/camera_python/nsCamera/utils/FlatField.py File Reference

Namespaces

- namespace [nsCamera](#)

- namespace [nsCamera.utils](#)
- namespace [nsCamera.utils.FlatField](#)

Functions

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

7.44 FlatField.py

[Go to the documentation of this file.](#)

```
00001 # -*- coding: utf-8 -*-
00002 """
00003 Functions for batch flat-field image corrections
00004
00005 ***Do not use this file as a template for new code development***
00006
00007 Author: Jeremy Martin Hill (jerhill@llnl.gov)
00008 Author: Matthew Dayton (dayton5@llnl.gov)
00009
00010 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00011 LLNL-CODE-838080
00012
00013 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00014 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00015 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00016 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00017 be made under this license.
00018
00019 Version: 2.1.2 (February 2025)
00020 """
00021
00022 import os
00023 import re
00024
00025 import numpy as np
00026 from PIL import Image
00027 from joblib import parallel, delayed
00028 from scipy.stats import theilslopes
00029 from skimage.external.tifffile import imread
```

```

00030
00031
00032 def getFilenames(frame="Frame 1"):
00033     """
00034     get a list of tiff filenames in current working director for frame
00035     """
00036     onlyfiles = next(os.walk("./"))[2]
00037     return [k for k in onlyfiles if frame in k and "tif" in k]
00038
00039
00040 def getROIvector(imgfilename, roi):
00041     """
00042     return a numpy row vector of version of the image
00043     """
00044     img = imread(imgfilename)
00045     vroi = img[roi[1] : (roi[3]), (roi[0] : (roi[2]))].flattenlist()
00046     return vroi
00047
00048
00049 def tslopes(x, y):
00050     """
00051     theilslopes implements a method for robust linear regression.
00052     It computes the slope as the median of all slopes between paired values.
00053     """
00054     val = theilslopes(x, y)
00055     return [val[0], val[1]]
00056
00057
00058 def generateFF(
00059     FRAMES=["Frame_0", "Frame_1", "Frame_2", "Frame_3"],
00060     roi=[0, 0, 512, 1024],
00061     directory="",
00062     ncores=-1,
00063 ):
00064     # TODO: documentation
00065     # use of ROI here not compatible with use of ROI in removeFF
00066
00067     if directory:
00068         cwd = os.getcwd()
00069         newpath = os.path.join(cwd, directory)
00070         os.chdir(newpath)
00071     if not FRAMES:
00072         print("No framelist provided, defaulting to four frames")
00073         FRAMES = ["Frame_0", "Frame_1", "Frame_2", "Frame_3"]
00074     for f in FRAMES:
00075         files = getFilenames(frame=f)
00076         imgslist = [getROIvector(fn, roi) for fn in files] # a list of flattened images
00077         imgsarray = np.vstack(imgslist) # turn the list into an array
00078         npix = np.shape(imgsarray)[1] # total number of pixels
00079         x = np.median(imgsarray, axis=1) # median of each image used for flat fielding
00080         y = []
00081         for i in range(npix):
00082             # each member of y represents a pixel, as a list of magnitudes over all the
00083             # images
00084             y.append(imgsarray[:, i])
00085         # get pixel gain and offset for flatfield ff using Thiel-Sen slopes
00086         ff = []
00087         ff = parallel.Parallel(n_jobs=ncores, verbose=5, pre_dispatch="2 * n_jobs")(
00088             delayed(tslopes)(x, pixel) for pixel in y
00089         )
00090         # x is the dependent variable; here uses median of image as characteristic of
00091         # noise level
00092         m, c = zip(*ff) # separate into gain and offset
00093         m = np.array(m)
00094         m[m < 0.1] = 0.1 # handle outliers
00095         m[m > 1000] = 1000 # handle outliers
00096         m = 1.0 / m
00097         m = m.reshape(roi[3] - roi[1], roi[2] - roi[0]) # turn into matrix
00098         c = np.array(c).reshape(roi[3] - roi[1], roi[2] - roi[0]) # turn into matrix
00099
00100         with open("px_gain_%s.txt" % f.replace("Frame_", "f"), "w+") as file:
00101             np.savetxt(file, m)
00102         with open("px_off_%s.txt" % f.replace("Frame_", "f"), "w+") as file:
00103             np.savetxt(file, c)
00104
00105
00106 def removeFF(filename, directory="", roi=[0, 0, 512, 1024]):
00107     if directory:
00108         cwd = os.getcwd()
00109         newpath = os.path.join(cwd, directory)
00110         os.chdir(newpath)

```

```

00111     framenum = re.search("Frame_(\d)", filename).group(1)
00112     gainFilename = "px_gain_f" + framenum + ".txt"
00113     gainall = np.loadtxt(gainFilename)
00114     gain = gainall[(roi[1]) : (roi[3]), (roi[0]) : (roi[2])]
00115     offFilename = "px_off_f" + framenum + ".txt"
00116     offsetall = np.loadtxt(offFilename, dtype="uint32")
00117     offset = offsetall[(roi[1]) : (roi[3]), (roi[0]) : (roi[2])]
00118
00119     beforeImageall = imread(filename)
00120     beforeImage = beforeImageall[(roi[1]) : (roi[3]), (roi[0]) : (roi[2])]
00121     imageMed = np.median(beforeImage)
00122
00123     flat = imageMed * gain + offset
00124     flat = flat.clip(0)
00125     fix = beforeImage - flat
00126     clipped = fix.clip(0)
00127     fixinit = clipped.astype("uint16")
00128     fixing = Image.fromarray(fixinit)
00129
00130     fixFilename = filename[:-4] + ".ff" + filename[-4:]
00131     fixing.save(fixFilename)
00132
00133 def removeFFall(
00134     directory="",
00135     FRAMES=["Frame_0", "Frame_1", "Frame_2", "Frame_3"],
00136     roi=[0, 0, 512, 1024],
00137 ):
00138     cwd = os.getcwd()
00139     if directory:
00140         newpath = os.path.join(cwd, directory)
00141     else:
00142         newpath = cwd
00143     os.chdir(newpath)
00144     files = next(os.walk("./"))[2]
00145     filelist = []
00146     for frame in FRAMES:
00147         filelist.extend([k for k in files if frame in k and ".tif" in k])
00148     for fname in filelist:
00149         removeFF(fname, directory, roi)
00150
00151
00152 if __name__ == "__main__":
00153     import argparse
00154
00155     parser = argparse.ArgumentParser()
00156     parser.add_argument(
00157         "-d", action="store", dest="directory", default="", help="VRST scan directory"
00158     )
00159     parser.add_argument(
00160         "-f",
00161         nargs="+",
00162         action="store",
00163         dest="frames",
00164         default="",
00165         help="Frame numbers to process, eg. -f 2 3",
00166     )
00167     args = parser.parse_args()
00168     framelist = ["Frame_" + str(frame) for frame in args.frames]
00169     generateFF(framelist, directory=args.directory)
00170
00171 """
00172 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00173 LLNL-CODE-838080
00174
00175 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00176 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00177 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00178 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00179 be made under this license.
00180 """

```


7.45 C:/Users/hill35/git/camera_python/nsCamera/utils/misc.py File Reference

Classes

- class [nsCamera.utils.misc.fakeCA](#)

Namespaces

- namespace [nsCamera](#)
- namespace [nsCamera.utils](#)
- namespace [nsCamera.utils.misc](#)

Functions

- [nsCamera.utils.misc.makeLogLabels](#) (logtag, label)
- [nsCamera.utils.misc.getEnter](#) (text)
- [nsCamera.utils.misc.checkCRC](#) (rval)
- [nsCamera.utils.misc.str2bytes](#) (astring)
- [nsCamera.utils.misc.bytes2str](#) (bytesequenece)
- [nsCamera.utils.misc.str2nparray](#) (valstring)
- [nsCamera.utils.misc.flattenList](#) (x)
- [nsCamera.utils.misc.generateFrames](#) (camassem, data, columns=1)
- [nsCamera.utils.misc.loadDumpedData](#) (filename="frames.txt", path=None, filetype="txt", sensor="daedalus", firstframe=None, lastframe=None, width=None, height=None, padToFull=None, firstrow=None, lastrow=None, maxwidth=None, maxheight=None, bytesperpixel=None, interlacing=None, columns=1)
- [nsCamera.utils.misc.saveTiffs](#) (self, frames, path=None, filename="Frame", prefix=None, index=None)
- [nsCamera.utils.misc.plotFrames](#) (self, frames, index=None)
- [nsCamera.utils.misc.partition](#) (self, frames, columns)

7.46 misc.py

[Go to the documentation of this file.](#)

```
00001 # -*- coding: utf-8 -*-
00002 """
00003 Miscellaneous utilities, including batch processing of images acquired using the
00004 nsCamera. These are functions that don't require a cameraAssembler object to be
00005 instantiated before use.
00006
00007 Author: Jeremy Martin Hill (jerhill@llnl.gov)
00008
00009 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00010 LLNL-CODE-838080
00011
00012 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00013 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00014 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00015 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00016 be made under this license.
00017
00018 Version: 2.1.2 (February 2025)
00019 """
00020
00021 import binascii
```

```

00022 import collections
00023 import logging
00024 import os
00025 import sys
00026 from datetime import datetime
00027
00028 import numpy as np
00029 from matplotlib import pyplot as plt
00030 from past.builtins import raw_input
00031 from PIL import Image
00032
00033 # TODO: is the self-reference necessary?
00034 # import nsCamera.utils.misc
00035 from nsCamera.utils import crc16pure
00036
00037
00038 # TODO: check error messages
00039 def makeLogLabels(logtag, label):
00040     if logtag is None:
00041         logtag = ""
00042
00043     logcritbase = "CRITICAL{logtag}: ".format(logtag=logtag)
00044     logerrbase = "ERROR{logtag}: ".format(logtag=logtag)
00045     logwarnbase = "WARNING{logtag}: ".format(logtag=logtag)
00046     loginfobase = "INFO{logtag}: ".format(logtag=logtag)
00047     logdebugbase = "DEBUG{logtag}: ".format(logtag=logtag)
00048
00049     logcrit = "{base}{label}".format(base=logcritbase, label=label)
00050     logerr = "{base}{label}".format(base=logerrbase, label=label)
00051     logwarn = "{base}{label}".format(base=logwarnbase, label=label)
00052     loginfo = "{base}{label}".format(base=loginfobase, label=label)
00053     logdebug = "{base}{label}".format(base=logdebugbase, label=label)
00054
00055     return logcrit, logerr, logwarn, loginfo, logdebug
00056
00057
00058 def getEnter(text):
00059     """
00060     Wait for enter key to be pressed.
00061
00062     Args:
00063         text: message asking for keypress
00064     """
00065     python, _, _, _, _ = sys.version_info
00066     if python >= 3:
00067         input(text)
00068     else:
00069         raw_input(text)
00070
00071
00072 def checkCRC(rval):
00073     """
00074     Calculate CRC for rval[:-4] and compare with expected CRC in rval[-4:]
00075
00076     Args:
00077         rval: hexadecimal string
00078
00079     Returns:
00080         boolean, True if CRCs match, False if they don't match or the input is invalid
00081     """
00082     if not isinstance(rval, str) or len(rval) < 5:
00083         logging.error("ERROR: checkCRC: Invalid input: {rval}".format(rval=rval))
00084         return False
00085     data_crc = int(rval[:-4], base=16)
00086     CRC_calc = crc16pure.crc16xmodem(str2bytes(rval[:-4]))
00087     return CRC_calc == data_crc
00088
00089
00090 def str2bytes(astring):
00091     """
00092     Python-version-agnostic converter of hexadecimal strings to bytes
00093
00094     Args:
00095         astring: hexadecimal string without '0x'
00096
00097     Returns:
00098         byte string equivalent to input string
00099     """
00100
00101     python, _, _, _, _ = sys.version_info
00102     if python >= 3:

```

```

00103         try:
00104             dbytes = binascii.a2b_hex(astring)
00105         except:
00106             logging.error(
00107                 "ERROR: str2bytes: invalid input: '{astring}'; returning zero"
00108                 " byte".format(astring=astring)
00109             )
00110             dbytes = b"\x00"
00111     else:
00112         try:
00113             dbytes = astring.decode("hex")
00114         except:
00115             logging.error(
00116                 "ERROR: str2bytes: invalid input: '{astring}'; returning zero "
00117                 " byte".format(astring=astring)
00118             )
00119             dbytes = b"\x00"
00120     return dbytes
00121
00122
00123 def bytes2str(bytessequence):
00124     """
00125     Python-version-agnostic converter of bytes to hexadecimal strings
00126
00127     Args:
00128         bytessequence: sequence of bytes as string (Py2) or bytes (Py3)
00129
00130     Returns:
00131         hexadecimal string representation of 'bytes' without '0x'
00132     """
00133     try:
00134         estr = binascii.b2a_hex(bytessequence)
00135     except TypeError:
00136         logging.error(
00137             "ERROR: bytes2str: Invalid byte sequence: '{bytessequence}'; returning an "
00138             " empty string".format(bytessequence=bytessequence)
00139         )
00140     return ""
00141
00142     python, _, _, _, _ = sys.version_info
00143     if python >= 3:
00144         estr = str(estr)[2:-1]
00145     return estr
00146
00147 def str2nparray(valstring):
00148     """
00149     Convert string into array of uint16s
00150
00151     Args:
00152         valstring: string of hexadecimal characters
00153
00154     Returns:
00155         numpy array of uint16
00156     """
00157     if not isinstance(valstring, str):
00158         logging.error(
00159             "ERROR: str2nparray: Invalid input: {valstring} is not a string. Returning "
00160             " an empty array".format(valstring=valstring)
00161         )
00162     return np.array([])
00163     stringlen = len(valstring)
00164     arraylen = int(stringlen / 4)
00165     outarray = np.empty(int(arraylen), dtype="uint16")
00166
00167     for i in range(0, arraylen):
00168         try:
00169             outarray[i] = int(valstring[4 * i : 4 * i + 4], 16)
00170         except ValueError:
00171             logging.error(
00172                 "ERROR: str2nparray: input string does not represent a hexadecimal "
00173                 " integer. Returning an empty array"
00174             )
00175     return np.array([])
00176     return outarray
00177
00178
00179 def flattenlist(x):
00180     """
00181     Flatten list of lists recursively into single list
00182     """
00183     python, _, _, _, _ = sys.version_info

```

```

00184     try:
00185         if python >= 3:
00186             if isinstance(x, collections.abc.Iterable):
00187                 return [a for i in x for a in flattenlist(i)]
00188             else:
00189                 return [x]
00190         else:
00191             if isinstance(x, collections.Iterable):
00192                 return [a for i in x for a in flattenlist(i)]
00193             else:
00194                 return [x]
00195     except RecursionError:
00196         logging.error(
00197             "ERROR: flattenlist: input '{x}' is pathological and cannot be flattened."
00198             " Attempting to return the input unchanged"
00199         )
00200     return x
00201
00202
00203 class fakeCA:
00204     """
00205     Fake 'cameraAssembler' object to use as a parameter object in offline functions.
00206     Returned by generateFrames(), it contains the frame details required to save and
00207     plot images.
00208     """
00209
00210     def __init__(
00211         self,
00212         sensorname="icarus2",
00213         firstframe=0,
00214         lastframe=3,
00215         width=512,
00216         height=1024,
00217         padToFull=True,
00218         firstrow=0,
00219         lastrow=1023,
00220         maxwidth=512,
00221         maxheight=1024,
00222         bytesperpixel=2,
00223         interlacing=None,
00224         columns=1,
00225         logtag=None,
00226     ):
00227         self.sensorname = sensorname
00228         self.boardname = None
00229         self.padToFull = padToFull
00230         if logtag is None:
00231             self.logtag = ""
00232         else:
00233             self.logtag = logtag
00234
00235         self.logcritbase = "CRITICAL" + self.logtag + ": "
00236         self.logerrbase = "ERROR" + self.logtag + ": "
00237         self.logwarnbase = "WARNING" + self.logtag + ": "
00238         self.loginfobase = "INFO" + self.logtag + ": "
00239         self.logdebugbase = "DEBUG" + self.logtag + ": "
00240
00241         self.logcrit = self.logcritbase + "[FS] "
00242         self.logerr = self.logerrbase + "[FS] "
00243         self.logwarn = self.logwarnbase + "[FS] "
00244         self.loginfo = self.loginfobase + "[FS] "
00245         self.logdebug = self.logdebugbase + "[FS] "
00246
00247         if self.sensorname == "icarus":
00248             import nsCamera.sensors.icarus as snsr
00249         elif self.sensorname == "icarus2":
00250             import nsCamera.sensors.icarus2 as snsr
00251         elif self.sensorname == "daedalus":
00252             import nsCamera.sensors.daedalus as snsr
00253
00254         self.sensor = snsr(self)
00255
00256     def partition(self, parsed, columns):
00257         # return nsCamera.utils.misc.partition(self, parsed, columns)
00258         return partition(self, parsed, columns)
00259
00260
00261 def generateFrames(camassem, data, columns=1):
00262     """
00263     Processes data stream from board into frames and applies sensor-specific parsing.
00264     Generates padded data for full-size option of setRows.

```

```

00265     If used for offline processing, replace the 'self' object with the parameter object
00266     returned by loadDumpedData().
00267     If the data stream is incomplete (e.g., from an interrupted download), the data is
00268     padded with zeros to the correct length.
00269
00270     Args:
00271         self: during normal operation, is the CameraAssembler object. During offline
00272               operation, is a parameters object as returned by loadDumpedData()
00273         data: text stream from board or loaded file, or numpy from loaded file
00274         columns: 1 for single image, 2 for separate hemisphere images
00275
00276     Returns: list of parsed frames
00277     """
00278     logging.debug("DEBUG" + camassem.logtag + ": generateFrames")
00279     if isinstance(data[0], str):
00280         allframes = str2npparray(data)
00281     else:
00282         allframes = data
00283     nframes = camassem.sensor.lastframe - camassem.sensor.firstframe + 1
00284     frames = [0] * nframes
00285     framesize = camassem.sensor.width * (
00286         camassem.sensor.lastrow - camassem.sensor.firstrow + 1
00287     )
00288     if hasattr(camassem, "ca"):
00289         padIt = camassem.ca.padToFull
00290     else:
00291         padIt = camassem.padToFull
00292     if padIt:
00293         toprows = camassem.sensor.firstrow
00294         botrows = (camassem.sensor.maxheight - 1) - camassem.sensor.lastrow
00295         padtop = np.zeros(toprows * camassem.sensor.maxwidth, dtype=int)
00296         padbot = np.zeros(botrows * camassem.sensor.maxwidth, dtype=int)
00297         for n in range(nframes):
00298             thisframe = np.concatenate(
00299                 (padtop, allframes[n * framesize : (n + 1) * framesize], padbot)
00300             )
00301             frames[n] = thisframe
00302     else:
00303         for n in range(nframes):
00304             frames[n] = allframes[n * framesize : (n + 1) * framesize]
00305     # self.clearStatus()
00306     parsed = camassem.sensor.parseReadoff(frames, columns)
00307     return parsed
00308
00309
00310 def loadDumpedData(
00311     filename="frames.txt",
00312     path=None,
00313     filetype="txt",
00314     sensor="daedalus",
00315     firstframe=None,
00316     lastframe=None,
00317     width=None,
00318     height=None,
00319     padToFull=None,
00320     firstrow=None,
00321     lastrow=None,
00322     maxwidth=None,
00323     maxheight=None,
00324     bytesperpixel=None,
00325     interlacing=None,
00326     columns=1,
00327 ):
00328     """_summary_
00329
00330     Output can be piped directly to saveTiffs:
00331         pars,frames=loadDumpedData(filename="Dump.npy")
00332         saveTiffs(pars,frames)
00333
00334     Args:
00335         filename (str, optional): _description_. Defaults to "frames.txt".
00336         path (_type_, optional): _description_. Defaults to None.
00337         filetype (str, optional): _description_. Defaults to "txt".
00338         sensor (str, optional): _description_. Defaults to "daedalus".
00339         firstframe (_type_, optional): _description_. Defaults to None.
00340         lastframe (_type_, optional): _description_. Defaults to None.
00341         width (_type_, optional): _description_. Defaults to None.
00342         height (_type_, optional): _description_. Defaults to None.
00343         padToFull (_type_, optional): _description_. Defaults to None.
00344         firstrow (_type_, optional): _description_. Defaults to None.
00345         lastrow (_type_, optional): _description_. Defaults to None.
00346         maxwidth (_type_, optional): _description_. Defaults to None.

```

```

00346         maxheight (_type_, optional): _description_. Defaults to None.
00347         interlacing (_type_, optional): _description_. Defaults to None.
00348         columns (int, optional): _description_. Defaults to 1.
00349
00350
00351     Returns:
00352         Tuple (parameter object, list of data frames)
00353     """
00354     logging.debug("DEBUG: loadDumpedData")
00355     if sensor.lower() == "daedalus":
00356         import nsCamera.sensors.daedalus as snsr
00357     elif sensor.lower() == "icarus":
00358         import nsCamera.sensors.icarus as snsr
00359     elif sensor.lower() == "icarus2":
00360         import nsCamera.sensors.icarus2 as snsr
00361     else:
00362         logging.error(
00363             "ERROR loadDumpedData: invalid sensor type provided, defaulting to icarus2"
00364         )
00365         import nsCamera.sensors.icarus2 as snsr
00366
00367     def buildEmptyFrames():
00368         cols = [0] * 512
00369         frame = np.array([cols] * (lastrow - firstrow + 1))
00370         return [frame] * (lastframe - firstframe + 1)
00371
00372     # get defaults from class declarations if not specified as parameter
00373     if firstframe is None:
00374         firstframe = snsr.firstframe
00375     if lastframe is None:
00376         lastframe = snsr.lastframe
00377     # TODO: first frame number
00378     if width is None:
00379         width = snsr.width
00380     if height is None:
00381         height = snsr.height
00382     if firstrow is None:
00383         firstrow = snsr.firstrow
00384     if lastrow is None:
00385         lastrow = snsr.lastrow
00386     if maxwidth is None:
00387         maxwidth = snsr.maxwidth
00388     if maxheight is None:
00389         maxheight = snsr.maxheight
00390     if bytesperpixel is None:
00391         bytesperpixel = snsr.bytesperpixel
00392     if interlacing is None:
00393         interlacing = snsr.interlacing
00394
00395     parameters = fakeCA(
00396         sensor,
00397         firstframe,
00398         lastframe,
00399         width,
00400         height,
00401         padToFull,
00402         firstrow,
00403         lastrow,
00404         maxwidth,
00405         maxheight,
00406         bytesperpixel,
00407         interlacing,
00408     )
00409
00410     if path is None:
00411         path = os.path.join(os.getcwd())
00412     datafile = os.path.join(path, filename)
00413     if filename[-3:].lower() == "txt":
00414         filetype = "txt"
00415     elif filename[-3:].lower() == "npz":
00416         filetype = "npz"
00417     # TODO: return empty frames if error
00418
00419     if filetype == "npz":
00420         expectedlength = (lastframe - firstframe + 1) * (lastrow - firstrow + 1) * width
00421         try:
00422             f = np.load(datafile)
00423             padding = expectedlength - len(f)
00424             if padding:
00425                 logging.warning(
00426                     "{logwarn}loadDumpedData: Payload is shorter than expected."

```

```

00427         " Padding with '0's".format(logwarn=parameters.logwarn)
00428     )
00429     f = np.pad(f, (0, padding), "constant", constant_values=(0))
00430     frames = generateFrames(parameters, f, columns)
00431     return parameters, frames
00432
00433 except OSError as err:
00434     logging.error(
00435         "{logerr}loadDumpedData: OS error: {err}. Returning empty"
00436         " frames.".format(logerr=parameters.logerr, err=err)
00437     )
00438     return parameters, buildEmptyFrames()
00439 except:
00440     logging.error(
00441         "{logerr}loadDumpedData: Unexpected error: {err}. Returning empty"
00442         " frames.".format(logerr=parameters.logerr, err=str(sys.exc_info()[0]))
00443     )
00444     return parameters, buildEmptyFrames()
00445 # if filetype is not explicitly npy, try loading as text
00446 else:
00447     # Payload size as string implied by provided parameters
00448     expectedlength = (
00449         4 * (lastframe - firstframe + 1) * (lastrow - firstrow + 1) * width
00450     )
00451
00452     try:
00453         f = open(datafile, "r")
00454         s = f.read()
00455
00456         padding = expectedlength - len(s)
00457         if padding:
00458             logging.warning(
00459                 "{logwarn}loadDumpedData: Payload is shorter than expected."
00460                 " Padding with '0's".format(logwarn=parameters.logwarn)
00461             )
00462             s = s.ljust(expectedlength, "0")
00463
00464         frames = generateFrames(parameters, s)
00465         return parameters, frames
00466
00467 except OSError as err:
00468     logging.error(
00469         "{logerr}loadDumpedData: OS error: {err}. Returning empty"
00470         " frames.".format(logerr=parameters.logerr, err=err)
00471     )
00472     return parameters, buildEmptyFrames()
00473 except ValueError:
00474     logging.error(
00475         "{logerr}loadDumpedData: Could not convert data to an integer."
00476         " Returning empty frames.".format(logerr=parameters.logerr)
00477     )
00478     return parameters, buildEmptyFrames()
00479 except:
00480     logging.error(
00481         "{logerr}loadDumpedData: Unexpected error: {err}. Returning empty"
00482         " frames.".format(logerr=parameters.logerr, err=str(sys.exc_info()[0]))
00483     )
00484     return parameters, buildEmptyFrames()
00485
00486
00487 def saveTiffs(
00488     self,
00489     frames,
00490     path=None,
00491     filename="Frame",
00492     prefix=None,
00493     index=None,
00494 ):
00495     """
00496     Save numpy array or list of numpy arrays or single array to disk as individual
00497     tiffs, with frame number appended to filename. If used for standalone, use the
00498     parameter object returned by loadDumpedData() as the first argument
00499
00500     Args:
00501         self: during normal operation, is cameraAssembler object. During offline
00502             operation, is the parameter object returned by loadDumpedData()
00503         frames: numpy array or list of numpy arrays
00504         path: save path, defaults to './output'
00505         filename: defaults to 'Frame' followed by frame number
00506         prefix: prepended to 'filename', defaults to time/date
00507             (e.g. '160830-124704_')

```

```

00508         index: number to start frame numbering
00509
00510     Returns:
00511         Error string
00512     """
00513     # logging.info("INFO" + self.logtag + ": saveTiffs")
00514     logging.info("{loginfo}: saveTiffs".format(loginfo=self.loginfo))
00515     err = ""
00516     if path is None:
00517         path = os.path.join(os.getcwd(), "output")
00518     if prefix is None:
00519         prefix = datetime.now().strftime("%y%m%d-%H%M%S%f")[:-5] + "_"
00520     if not os.path.exists(path):
00521         os.makedirs(path)
00522     if index is None:
00523         firstnum = self.sensor.firstframe
00524     else:
00525         firstnum = index
00526
00527     # if this is a text string from fast readoff, do the numpy conversion now
00528     if isinstance(frames[0], str):
00529         frames = generateFrames(frames)
00530
00531     frametemp = np.copy(frames)
00532     if np.issubdtype(type(frametemp[0]), np.number):
00533         # if type(frametemp[0]) is np.uint16:
00534         # single frame needs to be a list containing one frame
00535         frametemp = [frametemp]
00536     for idx, frame in enumerate(frametemp):
00537         if idx < len(frametemp) / 2:
00538             interlacing = self.sensor.interlacing[0]
00539         else:
00540             interlacing = self.sensor.interlacing[1]
00541         try:
00542             if self.padToFull:
00543                 frame = np.reshape(
00544                     frame, (self.sensor.maxheight // (interlacing + 1), -1)
00545                 )
00546             else:
00547                 frame = np.reshape(
00548                     frame,
00549                     (
00550                         (self.sensor.lastrow - self.sensor.firstrow + 1)
00551                         // (interlacing + 1),
00552                         -1,
00553                     ),
00554                 )
00555             frameimg = Image.fromarray(frame, "I;16")
00556             namenum = filename + "_%d" % firstnum
00557             tifpath = os.path.join(path, prefix + namenum + ".tif")
00558             frameimg.save(tifpath)
00559             firstnum += 1
00560         except Exception:
00561             err = "saveTiffs: unable to save images"
00562             # logging.error("ERROR" + self.logtag + ": " + err)
00563             logging.error("{logerr}: {err}".format(logerr=self.logerr, err=err))
00564     return err
00565
00566
00567 def plotFrames(self, frames, index=None):
00568     """
00569     Plot frame or list of frames as individual graphs.
00570
00571     Args:
00572         self: during normal operation, is cameraAssembler object. During offline
00573             operation, is the parameter object returned by loadDumpedData()
00574         frames: numpy array or list of numpy arrays
00575         index: number to start frame numbering
00576
00577     Returns:
00578         Error string
00579     """
00580     # logging.info(self.loginfo + "plotFrames: index = " + str(index))
00581     logging.info(
00582         "{loginfo}: plotFrames: index = {index}".format(
00583             loginfo=self.loginfo, index=index
00584         )
00585     )
00586     err = ""
00587     if index is None:
00588         nframe = self.sensor.firstframe

```



```

00589     else:
00590         nframe = index
00591
00592     if not isinstance(frames, list):
00593         frames = [frames]
00594
00595     # if this is a text string from fast readoff, do the numpy conversion now
00596     if isinstance(frames[0], str):
00597         frames = generateFrames(frames)
00598
00599     framestemp = np.copy(frames)
00600     for idx, frame in enumerate(framestemp):
00601         if idx < len(framestemp) / 2:
00602             interlacing = self.sensor.interlacing[0]
00603         else:
00604             interlacing = self.sensor.interlacing[1]
00605         try:
00606             if self.padToFull:
00607                 frame = np.reshape(
00608                     frame, (self.sensor.maxheight // (interlacing + 1), -1)
00609                 )
00610             else:
00611                 frame = np.reshape(
00612                     frame,
00613                     (
00614                         (self.sensor.lastrow - self.sensor.firstrow + 1)
00615                         // (interlacing + 1),
00616                         -1,
00617                     ),
00618                 )
00619         except:
00620             err = "{logerr}plotFrames: unable to plot frame".format(logerr=self.logerr)
00621             logging.error(err)
00622             continue
00623         plt.imshow(frame, cmap="gray")
00624         name = "Frame %d" % nframe
00625         plt.title(name)
00626         plt.show()
00627         nframe += 1
00628     return err
00629
00630
00631 # TODO: separate images for hemispheres with different timing
00632
00633
00634 def partition(self, frames, columns):
00635     """
00636     Extracts interlaced frames and divides images by hemispheres. If interlacing does
00637     not evenly divide the height, remainder lines will be dropped
00638
00639     Args:
00640         self: during normal operation, is sensor object. During offline
00641             operation, is the parameter.sensor object returned by loadDumpedData()
00642         frames: list of full-sized frames
00643         columns: 1 for single image, 2 for separate hemisphere images
00644
00645     Returns: list of deinterlaced frames
00646     """
00647     logging.debug(
00648         "{logdebug}partition: columns = {columns}, interlacing = {interlacing}".format(
00649             logdebug=self.logdebug, columns=columns, interlacing=self.sensor.interlacing
00650         )
00651     )
00652
00653     def unshuffle(frames, ifactor):
00654         warntrimmed = False
00655         if self.padToFull:
00656             newheight = self.sensor.maxheight // (ifactor + 1)
00657             if newheight != (self.sensor.maxheight / (ifactor + 1)):
00658                 warntrimmed = True
00659         else:
00660             newheight = self.sensor.height // (ifactor + 1)
00661             if newheight != (self.sensor.height / (ifactor + 1)):
00662                 warntrimmed = True
00663
00664         if warntrimmed:
00665             logging.warning(
00666                 "{logwarn} partition: interlacing setting requires dropping of lines to"
00667                 " maintain consistent frame sizes ".format(logwarn=self.logwarn)
00668             )
00669         delaced = []

```

```

00670         for frame in frames:
00671             for sub in range(ifactor + 1):
00672                 current = np.zeros((newheight, self.sensor.width // columns), dtype=int)
00673                 for line in range(newheight):
00674                     current[line] = frame[(ifactor + 1) * line + sub]
00675                 delaced.append(current)
00676             nframes = self.sensor.lastframe - self.sensor.firstframe + 1
00677             resorted = [None] * len(delaced)
00678             for sub in range(ifactor + 1):
00679                 for idx, frame in enumerate(frames):
00680                     resorted[sub * nframes + idx] = delaced[idx * (ifactor + 1) + sub]
00681             return resorted
00682
00683     if self.sensor.interlacing[0] != self.sensor.interlacing[1]:
00684         columns = 2 # true even if not explicitly requested by readoff
00685     if columns == 1:
00686         if self.sensor.interlacing == [0, 0]: # don't do anything
00687             return frames
00688         else:
00689             return unshuffle(frames, self.sensor.interlacing[0])
00690     else:
00691         # reshape frame into the proper shape, then split horizontally
00692         if self.padToFull:
00693             framesab = [
00694                 np.hsplit(frame.reshape(self.sensor.maxheight, -1), 2)
00695                 for frame in frames
00696             ]
00697         else:
00698             framesab = [
00699                 np.hsplit(
00700                     frame.reshape((self.sensor.lastrow - self.sensor.firstrow + 1), -1),
00701                     2,
00702                 )
00703                 for frame in frames
00704             ]
00705         framesa = [hemis[0] for hemis in framesab]
00706         framesb = [hemis[1] for hemis in framesab]
00707     if self.sensor.interlacing == [0, 0]:
00708         return framesa + framesb
00709     else:
00710         return unshuffle(framesa, self.sensor.interlacing[0]) + unshuffle(
00711             framesb, self.sensor.interlacing[1]
00712         )
00713
00714
00715 """
00716 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00717 LLNL-CODE-838080
00718
00719 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00720 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00721 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00722 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00723 be made under this license.
00724 """

```

7.47 C:/Users/hill35/git/camera_python/nsCamera/utils/Packet.py File Reference

Classes

- class [nsCamera.utils.Packet.Packet](#)

Namespaces

- namespace [nsCamera](#)
- namespace [nsCamera.utils](#)
- namespace [nsCamera.utils.Packet](#)

7.48 Packet.py

[Go to the documentation of this file.](#)

```

00001 # -*- coding: utf-8 -*-
00002 """
00003 Packet object for communication with boards
00004
00005 Author: Brad Funsten (funsten1@llnl.gov)
00006 Author: Jeremy Hill (hill35@llnl.gov)
00007
00008 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00009 LLNL-CODE-838080
00010
00011 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00012 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00013 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00014 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00015 be made under this license.
00016
00017 Version: 2.1.2 (February 2025)
00018 """
00019
00020 from __future__ import absolute_import
00021
00022 # mport binascii
00023 import sys
00024
00025 from nsCamera.utils import crc16pure
00026 from nsCamera.utils.misc import str2bytes
00027
00028
00029 class Packet:
00030     """
00031     Packet object for communication with boards. See ICD for details.
00032
00033     Single Command/Response packet:
00034     +-----+-----+-----+-----+-----+
00035     | 16 bits | 4 bits | 12 bits | 32 bits | 16 bits |
00036     | Preamble | Command | Address | Data | CRC16 |
00037     +-----+-----+-----+-----+-----+
00038
00039     Read Burst Response packet:
00040
00041     +-----+-----+-----+-----+
00042     | 16 bits | 4 bits | 4 bits | 16 bits | %
00043     | Preamble | Command | Sub-command | Sequence ID | %
00044     +-----+-----+-----+-----+
00045
00046     +-----+-----+-----+
00047     % 16 bits | Variable | 16 bits |
00048     % Payload Length | Payload | CRC16 |
00049     +-----+-----+-----+
00050
00051     """
00052     def __init__(
00053         # NOTE: 'numerical' components are handled as hex strings
00054         self,
00055         preamble="aaaa",
00056         cmd="0",
00057         addr="",
00058         data="00000000",
00059         seqID="",
00060         payload_length="",
00061         payload="",
00062         crc="",
00063     ):
00064         self.PY3 = sys.version_info > (3,)
00065         self.preamble = preamble # 16 bit packet preamble
00066         self.cmd = str(cmd) # 4 bit command packet
00067         self.addr = addr.zfill(3) # 12 bit address packet
00068         self.data = data.zfill(8) # 32 bit data packet
00069         # 16 bit sequence ID packet (only Read Burst)
00070         self.seqID = seqID
00071         # 16 bit payload packet (only Read Burst)
00072         self.payload_length = payload_length
00073         # variable payload packet (only Read Burst) for now it's 16 bits
00074         self.payload = payload
00075         # 16 bit CRC-CCIT (XModem) packet
00076         self.crc = crc

```

```

00077         self.type = ""
00078         if self.crc == "": # check if packet to be sent needs crc appended
00079             self.crc = self.calculateCRC()
00080
00081     def pktStr(self):
00082         """
00083         Generate hexadecimal string form of packet
00084
00085         Returns:
00086             packet as hexadecimal string without '0x'
00087         """
00088         if self.seqID != "":
00089             # Read burst response
00090             packetparts = [
00091                 self.preamble,
00092                 self.cmd,
00093                 self.seqID,
00094                 self.payload_length,
00095                 self.payload,
00096                 self.crc,
00097             ]
00098         else:
00099             # Single Command/Response response
00100             packetparts = [self.preamble, self.cmd, self.addr, self.data, self.crc]
00101         stringparts = [
00102             part.decode("ascii") if isinstance(part, bytes) else part
00103             for part in packetparts
00104         ]
00105         out = "".join(stringparts)
00106         return out
00107
00108     def calculateCRC(self):
00109         """
00110         Calculate CRC-CCIT (XModem) (2 bytes) from 8 byte packet for send and rcv
00111
00112         Returns:
00113             CRC as hexadecimal string without '0x'
00114         """
00115         preamble = self.preamble
00116         crc = self.crc
00117         self.crc = ""
00118         self.preamble = ""
00119
00120         CRC_dec = crc16pure.crc16xmodem(str2bytes(self.pktStr()))
00121         # input = int type decimal, output = hex string with 0x at the beginning
00122         CRC_hex_0x = "0x%0.4X" % CRC_dec
00123         # make all hex letters lower case for comparison
00124         CRC_hex = CRC_hex_0x.lower()
00125         # input = hex string with 0x at the beginning, output = hex str with 0x removed
00126         CRC_hex = CRC_hex[2:]
00127         self.preamble = preamble
00128         self.crc = crc
00129         return CRC_hex
00130
00131     def checkCRC(self):
00132         """
00133         Returns: boolean, True if CRC check passes
00134         """
00135         return self.calculateCRC() == self.crc
00136
00137     def checkReadPacket(self, resppkt):
00138         """
00139         Confirm that Read Single occurred without error
00140         Args:
00141             resppkt: response packet
00142
00143         Returns:
00144             tuple (error string, response packet as string)
00145         """
00146         err = ""
00147         if int(resp_pkt.cmd.upper(), 16) - int(self.cmd.upper(), 16) != 0x8:
00148             err = "invalid command; "
00149         if resp_pkt.addr.upper() != self.addr.upper():
00150             err += "invalid address; "
00151         if resp_pkt.crc.upper() != resp_pkt.calculateCRC().upper():
00152             err += "invalid CRC; "
00153         return err, resp_pkt.pktStr()
00154
00155     def checkResponsePacket(self, resp_pkt):
00156         """
00157         Confirm that Write Single occurred without error

```

```

00158     Args:
00159         resppkt: response packet
00160
00161     Returns:
00162         tuple (error string, response packet as string)
00163     """
00164     err = ""
00165     if int(resppkt.data, 16) & 1:
00166         err += "Checksum error; "
00167     if int(resppkt.data, 16) & 2:
00168         err += "Invalid command / command not executed; "
00169     err1, rval = self.checkReadPacket(resppkt)
00170     err += err1
00171     return err, rval
00172
00173 def checkResponseString(self, respstr):
00174     """
00175     Checks response string for error indicators
00176     Args:
00177         respstr: packet as hexadecimal string
00178
00179     Returns:
00180         tuple (error string, response packet string)
00181     """
00182     respstring = respstr.decode(encoding="UTF-8")
00183     resppkt = Packet(
00184         preamble=respstring[0:4],
00185         cmd=respstring[4],
00186         addr=respstring[5:8],
00187         data=respstring[8:16],
00188     )
00189
00190     if resppkt.cmd == "8":
00191         # verify response to write command
00192         err, rval = self.checkResponsePacket(resppkt)
00193     elif resppkt.cmd == "9":
00194         err, rval = self.checkReadPacket(resppkt) # verify response to read command
00195     else:
00196         err = "Packet command invalid; "
00197         rval = ""
00198     return err, rval
00199
00200
00201 """
00202 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00203 LLNL-CODE-838080
00204
00205 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00206 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00207 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00208 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00209 be made under this license.
00210 """

```

7.49 C:/Users/hill35/git/camera_python/nsCamera/utils/Subregister.py File Reference

Classes

- class [nsCamera.utils.Subregister.SubRegister](#)

Namespaces

- namespace [nsCamera](#)
- namespace [nsCamera.utils](#)
- namespace [nsCamera.utils.Subregister](#)

7.50 Subregister.py

[Go to the documentation of this file.](#)

```

00001 # -*- coding: utf-8 -*-
00002 """
00003 Subregister object represents a subset of a full register
00004
00005 Author: Matthew Dayton (dayton5@llnl.gov)
00006 Author: Jeremy Martin Hill (jerhill@llnl.gov)
00007
00008 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00009 LLNL-CODE-838080
00010
00011 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00012 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00013 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00014 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00015 be made under this license.
00016
00017 Version: 2.1.2 (February 2025)
00018 """
00019
00020
00021 class SubRegister:
00022     """
00023     Represents a subset of a 32-bit register [31..0] starting at 'start_bit' consisting
00024     of 'width' bits. Consistent with the ICD usage, start_bit is MSB e.g., for [7..0],
00025     the start_bit is '7'.
00026     """
00027
00028     def __init__(
00029         self,
00030         board,
00031         name,
00032         register,
00033         start_bit=31,
00034         width=8,
00035         writable=False,
00036         value=255,
00037         minV=0,
00038         maxV=5,
00039     ):
00040         self.name = name
00041         self.register = register
00042         self.addr = board.registers[register]
00043         self.start_bit = start_bit
00044         self.width = width
00045         self.value = value
00046         self.max_value = 2**width - 1 # used to normalize the input values to 1
00047         self.min = 0
00048         self.max = self.max_value
00049         self.writable = writable
00050         self.minV = minV
00051         self.maxV = maxV
00052         # resolution should be reset after init if actual min and max are different
00053         self.resolution = (1.0 * maxV - minV) / self.max_value
00054
00055     """
00056
00057 Copyright (c) 2025, Lawrence Livermore National Security, LLC. All rights reserved.
00058 LLNL-CODE-838080
00059
00060 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
00061 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy (DOE)
00062 and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
00063 'nsCamera' is distributed under the terms of the MIT license. All new contributions must
00064 be made under this license.
00065 """

```

Index

- `_CRT_SECURE_NO_WARNINGS`
 - Data.c, [304](#)
 - UPnP.c, [344](#)
- `__all__`
 - nsCamera, [10](#)
 - nsCamera.boards, [10](#)
 - nsCamera.comms, [13](#)
 - nsCamera.sensors, [14](#)
 - nsCamera.utils, [17](#)
- `__init__`
 - nsCamera.boards.LLNL_v1.IInl_v1, [153](#)
 - nsCamera.boards.LLNL_v4.IInl_v4, [173](#)
 - nsCamera.CameraAssembler.CameraAssembler, [43](#)
 - nsCamera.comms.GigE.GigE, [124](#)
 - nsCamera.comms.RS422.RS422, [199](#)
 - nsCamera.sensors.daedalus.daedalus, [99](#)
 - nsCamera.sensors.icarus.icarus, [139](#)
 - nsCamera.sensors.icarus2.icarus2, [146](#)
 - nsCamera.sensors.sensorBase.sensorBase, [212](#)
 - nsCamera.utils.misc.fakeCA, [119](#)
 - nsCamera.utils.Packet.Packet, [193](#)
 - nsCamera.utils.Subregister.SubRegister, [229](#)
- `_crc16`
 - nsCamera.utils.crc16pure, [18](#)
- `_fields_`
 - nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO, [232](#)
- `_ser`
 - nsCamera.comms.RS422.RS422, [208](#)
- `_strnicmp`
 - UPnP.c, [344](#)
- `_zest`
 - nsCamera.comms.GigE.GigE, [133](#)
- `abort`
 - nsCamera.CameraAssembler.CameraAssembler, [89](#)
- `abortReadoff`
 - nsCamera.CameraAssembler.CameraAssembler, [46](#)
- `action`
 - nsCamera.utils.FlatField, [22](#)
- `ADC5_bipolar`
 - nsCamera.boards.LLNL_v1.IInl_v1, [169](#)
 - nsCamera.boards.LLNL_v4.IInl_v4, [189](#)
- `ADC5_mult`
 - nsCamera.boards.LLNL_v1.IInl_v1, [169](#)
 - nsCamera.boards.LLNL_v4.IInl_v4, [189](#)
- `addr`
 - nsCamera.utils.Packet.Packet, [196](#)
 - nsCamera.utils.Subregister.SubRegister, [229](#)
- `arch`
 - nsCamera.CameraAssembler.CameraAssembler, [89](#)
- `args`
 - nsCamera.utils.FlatField, [22](#)
- `arm`
 - nsCamera.CameraAssembler.CameraAssembler, [47](#)
 - nsCamera.comms.GigE.GigE, [125](#)
 - nsCamera.comms.RS422.RS422, [201](#)
- `armed`
 - nsCamera.CameraAssembler.CameraAssembler, [89](#)
- `batchAcquire`
 - nsCamera.CameraAssembler.CameraAssembler, [47](#)
- `baud`
 - nsCamera.comms.RS422.RS422, [208](#)
- `board`
 - nsCamera.CameraAssembler.CameraAssembler, [89](#)
- `boardname`
 - nsCamera.CameraAssembler.CameraAssembler, [89](#)
 - nsCamera.utils.misc.fakeCA, [120](#)
- `bytes2str`
 - nsCamera.CameraAssembler.CameraAssembler, [48](#)
 - nsCamera.utils.misc, [24](#)
- `bytesperpixel`
 - nsCamera.sensors.daedalus.daedalus, [113](#)
 - nsCamera.sensors.icarus.icarus, [141](#)
 - nsCamera.sensors.icarus2.icarus2, [148](#)
- `C:/Users/hill35/git/camera_python/nsCamera/__init__.py,`
[233](#)
- `C:/Users/hill35/git/camera_python/nsCamera/boards/__init__.py,`
[234](#)
- `C:/Users/hill35/git/camera_python/nsCamera/boards/LLNL_v1.py,`
[237](#), [238](#)
- `C:/Users/hill35/git/camera_python/nsCamera/boards/LLNL_v4.py,`
[249](#)
- `C:/Users/hill35/git/camera_python/nsCamera/CameraAssembler.py,`
[260](#), [261](#)
- `C:/Users/hill35/git/camera_python/nsCamera/comms/__init__.py,`
[235](#)
- `C:/Users/hill35/git/camera_python/nsCamera/comms/GigE.py,`
[287](#)

- C:/Users/hill35/git/camera_python/nsCamera/comms/RS422.py, nsCamera.utils.Packet.Packet, 194
294
- C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/daedalus.py, nsCamera.utils.Packet.Packet, 194
301, 315
- C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/EndoCamera.CameraAssembler.CameraAssembler, 49
326, 327
- C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/Misc.py, nsCamera.utils.Packet.Packet, 194
329, 330
- C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/Private.h, nsCamera.utils.Packet.Packet, 195
331, 340
- C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/RS02Camera.CameraAssembler.CameraAssembler, 50
342, 354
- C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/ZestETM1.py, nsCamera.sensors.icarus.icarus, 140
362, 375
- C:/Users/hill35/git/camera_python/nsCamera/comms/ZestETM1/ZestETM1.py, nsCamera.sensors.sensorBase.sensorBase, 213
362, 375
- C:/Users/hill35/git/camera_python/nsCamera/sensors/__init__.py, nsCamera.boards.LLNL_v1.Llnl_v1, 155
235, 236
- C:/Users/hill35/git/camera_python/nsCamera/sensors/daedalus.py, nsCamera.boards.LLNL_v4.Llnl_v4, 175
377
- C:/Users/hill35/git/camera_python/nsCamera/sensors/daedalus.py, nsCamera.CameraAssembler.CameraAssembler, 50
377
- C:/Users/hill35/git/camera_python/nsCamera/sensors/icarus.py, nsCamera.boards.LLNL_v1.Llnl_v1, 155
387
- C:/Users/hill35/git/camera_python/nsCamera/sensors/icarus2.py, nsCamera.boards.LLNL_v4.Llnl_v4, 175
390
- C:/Users/hill35/git/camera_python/nsCamera/sensors/icarus2.py, nsCamera.CameraAssembler.CameraAssembler, 50
390
- C:/Users/hill35/git/camera_python/nsCamera/sensors/sensorBase.py, nsCamera.boards.LLNL_v1.Llnl_v1, 155
392
- C:/Users/hill35/git/camera_python/nsCamera/sensors/sensorBase.py, nsCamera.boards.LLNL_v4.Llnl_v4, 175
392
- C:/Users/hill35/git/camera_python/nsCamera/utils/__init__.py, nsCamera.CameraAssembler.CameraAssembler, 50
236, 237
- C:/Users/hill35/git/camera_python/nsCamera/utils/crc16pure.py, nsCamera.CameraAssembler.CameraAssembler, 50
401, 402
- C:/Users/hill35/git/camera_python/nsCamera/utils/crc16pure.py, nsCamera.comms.GigE.GigE, 134
401, 402
- C:/Users/hill35/git/camera_python/nsCamera/utils/FlatField.py, nsCamera.CameraAssembler.CameraAssembler, 50
405, 406
- C:/Users/hill35/git/camera_python/nsCamera/utils/FlatField.py, nsCamera.comms.GigE.GigE, 126
405, 406
- C:/Users/hill35/git/camera_python/nsCamera/utils/misc.py, nsCamera.comms.RS422.RS422, 201
409
- C:/Users/hill35/git/camera_python/nsCamera/utils/misc.py, closesocket
409
- C:/Users/hill35/git/camera_python/nsCamera/utils/Package.py, Private.h, 333
418, 419
- C:/Users/hill35/git/camera_python/nsCamera/utils/Package.py, cmd
418, 419
- C:/Users/hill35/git/camera_python/nsCamera/utils/Subregister.py, nsCamera.utils.Packet.Packet, 196
421, 422
- C:/Users/hill35/git/camera_python/nsCamera/utils/Subregister.py, columns
421, 422
- ca
nsCamera.sensors.daedalus.daedalus, 114
nsCamera.sensors.icarus.icarus, 141
nsCamera.sensors.icarus2.icarus2, 148
- calculateCRC
nsCamera.CameraAssembler.CameraAssembler, 89
nsCamera.CameraAssembler.CameraAssembler, 90
- CardInfo
nsCamera.configADCs
nsCamera.boards.LLNL_v1.Llnl_v1, 156
nsCamera.boards.LLNL_v4.Llnl_v4, 176
nsCamera.CameraAssembler.CameraAssembler, 51
- CardInfoP
nsCamera.Connection
nsCamera.comms.GigE.GigE, 134
- checkCRC
nsCamera.crc
nsCamera.utils.Packet.Packet, 197
- nsCamera.CRC16_XMODEM_TABLE
nsCamera.crc16pure, 18
- nsCamera.crc16xmodem

- nsCamera.utils.crc16pure, 18
- currttime
 - nsCamera.CameraAssembler.CameraAssembler, 90
- cycle
 - nsCamera.CameraAssembler.CameraAssembler, 90
- daedalus_monitor_controls
 - nsCamera.boards.LLNL_v1.IInl_v1, 169
 - nsCamera.boards.LLNL_v4.IInl_v4, 189
- daedalus_subreg_aliases
 - nsCamera.boards.LLNL_v1.IInl_v1, 169
 - nsCamera.boards.LLNL_v4.IInl_v4, 189
- data
 - nsCamera.utils.Packet.Packet, 197
- Data.c
 - _CRT_SECURE_NO_WARNINGS, 304
 - ZestETM1_CloseConnection, 305
 - ZESTETM1_COMMAND_MAILBOX_INT, 304
 - ZESTETM1_COMMAND_READ_REG, 304
 - ZESTETM1_COMMAND_SPI, 304
 - ZESTETM1_COMMAND_WRITE_REG, 304
 - ZestETM1_OpenConnection, 305
 - ZestETM1_ReadData, 307
 - ZestETM1_SendCommand, 308
 - ZestETM1_SPIReadWrite, 308
 - ZestETM1_WriteData, 309
 - ZestETM1CloseConnection, 311
 - ZestETM1OpenConnection, 311
 - ZestETM1ReadData, 311
 - ZestETM1ReadRegister, 312
 - ZestETM1SetInterrupt, 313
 - ZestETM1SPIReadWrite, 313
 - ZestETM1WriteData, 314
 - ZestETM1WriteRegister, 314
- datatimeout
 - nsCamera.comms.RS422.RS422, 208
- default
 - nsCamera.utils.FlatField, 22
- defoff
 - nsCamera.boards.LLNL_v4.IInl_v4, 189
- defsens
 - nsCamera.boards.LLNL_v4.IInl_v4, 189
- dest
 - nsCamera.utils.FlatField, 22
- detect
 - nsCamera.sensors.daedalus.daedalus, 114
 - nsCamera.sensors.icarus.icarus, 141
 - nsCamera.sensors.icarus2.icarus2, 148
- directory
 - nsCamera.utils.FlatField, 23
- disarm
 - nsCamera.boards.LLNL_v1.IInl_v1, 156
 - nsCamera.boards.LLNL_v4.IInl_v4, 176
 - nsCamera.CameraAssembler.CameraAssembler, 51
- dport
 - nsCamera.comms.GigE.GigE, 134
- dumpNumpy
 - nsCamera.CameraAssembler.CameraAssembler, 51
- dumpRegisters
 - nsCamera.CameraAssembler.CameraAssembler, 52
- dumpStatus
 - nsCamera.boards.LLNL_v1.IInl_v1, 157
 - nsCamera.boards.LLNL_v4.IInl_v4, 177
 - nsCamera.CameraAssembler.CameraAssembler, 53
- dumpSubregisters
 - nsCamera.CameraAssembler.CameraAssembler, 53
- enableLED
 - nsCamera.boards.LLNL_v1.IInl_v1, 158
 - nsCamera.boards.LLNL_v4.IInl_v4, 178
 - nsCamera.CameraAssembler.CameraAssembler, 53
- Error.c
 - ZestETM1_ErrorHandler, 327
 - ZestETM1_ErrorStrings, 327
 - ZestETM1GetErrorMessage, 326
 - ZestETM1RegisterErrorHandler, 326
- firstframe
 - nsCamera.sensors.daedalus.daedalus, 114
 - nsCamera.sensors.icarus.icarus, 141
 - nsCamera.sensors.icarus2.icarus2, 148
- firstrow
 - nsCamera.sensors.daedalus.daedalus, 114
 - nsCamera.sensors.icarus.icarus, 141
 - nsCamera.sensors.icarus2.icarus2, 148
- flattenlist
 - nsCamera.CameraAssembler.CameraAssembler, 54
 - nsCamera.utils.misc, 25
- FPGAboardtype
 - nsCamera.CameraAssembler.CameraAssembler, 90
- FPGAinterfaces
 - nsCamera.CameraAssembler.CameraAssembler, 90
- FPGAinvalid
 - nsCamera.CameraAssembler.CameraAssembler, 90
- FPGANum
 - nsCamera.CameraAssembler.CameraAssembler, 90
- fpganumID
 - nsCamera.sensors.daedalus.daedalus, 114
 - nsCamera.sensors.icarus.icarus, 142
 - nsCamera.sensors.icarus2.icarus2, 149
 - nsCamera.sensors.sensorBase.sensorBase, 227
- FPGArad
 - nsCamera.CameraAssembler.CameraAssembler, 91
- FPGAsensor
 - nsCamera.CameraAssembler.CameraAssembler, 91
- FPGAVersion
 - nsCamera.CameraAssembler.CameraAssembler, 91
- framelist
 - nsCamera.utils.FlatField, 23

- generateFF
 - nsCamera.utils.FlatField, 20
- generateFrames
 - nsCamera.utils.misc, 26
- getBoardInfo
 - nsCamera.CameraAssembler.CameraAssembler, 54
- getCardInfo
 - nsCamera.comms.GigE.GigE, 126
- getCardIP
 - nsCamera.comms.GigE.GigE, 127
- getEnter
 - nsCamera.CameraAssembler.CameraAssembler, 55
 - nsCamera.utils.misc, 27
- getFilenames
 - nsCamera.utils.FlatField, 20
- getManualTiming
 - nsCamera.CameraAssembler.CameraAssembler, 55
 - nsCamera.sensors.daedalus.daedalus, 100
 - nsCamera.sensors.sensorBase.sensorBase, 213
- getMonV
 - nsCamera.CameraAssembler.CameraAssembler, 55
- getPot
 - nsCamera.CameraAssembler.CameraAssembler, 56
- getPotV
 - nsCamera.CameraAssembler.CameraAssembler, 57
- getPressure
 - nsCamera.boards.LLNL_v1.IInI_v1, 159
 - nsCamera.boards.LLNL_v4.IInI_v4, 179
 - nsCamera.CameraAssembler.CameraAssembler, 58
- getRegister
 - nsCamera.CameraAssembler.CameraAssembler, 59
- getROIvector
 - nsCamera.utils.FlatField, 20
- getSensorStatus
 - nsCamera.sensors.sensorBase.sensorBase, 214
- getSensTemp
 - nsCamera.CameraAssembler.CameraAssembler, 59
 - nsCamera.sensors.daedalus.daedalus, 101
 - nsCamera.sensors.sensorBase.sensorBase, 214
- getSubregister
 - nsCamera.CameraAssembler.CameraAssembler, 60
- getTemp
 - nsCamera.boards.LLNL_v1.IInI_v1, 159
 - nsCamera.boards.LLNL_v4.IInI_v4, 180
 - nsCamera.CameraAssembler.CameraAssembler, 61
- getTimer
 - nsCamera.boards.LLNL_v1.IInI_v1, 160
 - nsCamera.boards.LLNL_v4.IInI_v4, 180
 - nsCamera.CameraAssembler.CameraAssembler, 61
- getTiming
 - nsCamera.CameraAssembler.CameraAssembler, 61
 - nsCamera.sensors.sensorBase.sensorBase, 214
- height
 - nsCamera.sensors.daedalus.daedalus, 114
 - nsCamera.sensors.icarus.icarus, 142
 - nsCamera.sensors.icarus2.icarus2, 149
- help
 - nsCamera.utils.FlatField, 23
- HFW
 - nsCamera.sensors.daedalus.daedalus, 115
- icarus_monitor_controls
 - nsCamera.boards.LLNL_v1.IInI_v1, 169
 - nsCamera.boards.LLNL_v4.IInI_v4, 189
- icarus_subreg_aliases
 - nsCamera.boards.LLNL_v1.IInI_v1, 169
 - nsCamera.boards.LLNL_v4.IInI_v4, 190
- icarustype
 - nsCamera.sensors.icarus.icarus, 142
 - nsCamera.sensors.icarus2.icarus2, 149
- init_board_specific
 - nsCamera.sensors.sensorBase.sensorBase, 216
- initBoard
 - nsCamera.boards.LLNL_v1.IInI_v1, 160
 - nsCamera.boards.LLNL_v4.IInI_v4, 181
 - nsCamera.CameraAssembler.CameraAssembler, 61
- initialize
 - nsCamera.CameraAssembler.CameraAssembler, 61
- initPots
 - nsCamera.boards.LLNL_v1.IInI_v1, 161
 - nsCamera.boards.LLNL_v4.IInI_v4, 181
 - nsCamera.CameraAssembler.CameraAssembler, 63
- initPowerCheck
 - nsCamera.CameraAssembler.CameraAssembler, 63
- initSensor
 - nsCamera.boards.LLNL_v1.IInI_v1, 162
 - nsCamera.boards.LLNL_v4.IInI_v4, 181
 - nsCamera.CameraAssembler.CameraAssembler, 64
- inittime
 - nsCamera.CameraAssembler.CameraAssembler, 91
- interlacing
 - nsCamera.sensors.daedalus.daedalus, 115
 - nsCamera.sensors.icarus.icarus, 142
 - nsCamera.sensors.icarus2.icarus2, 149
- iplist
 - nsCamera.CameraAssembler.CameraAssembler, 91
- lastframe
 - nsCamera.sensors.daedalus.daedalus, 115
 - nsCamera.sensors.icarus.icarus, 142
 - nsCamera.sensors.icarus2.icarus2, 149
- lastrow
 - nsCamera.sensors.daedalus.daedalus, 115
 - nsCamera.sensors.icarus.icarus, 142
 - nsCamera.sensors.icarus2.icarus2, 149
- latchPots
 - nsCamera.boards.LLNL_v1.IInI_v1, 162
 - nsCamera.boards.LLNL_v4.IInI_v4, 182

- nsCamera.CameraAssembler.CameraAssembler, 64
- loadDumpedData
 - nsCamera.utils.misc, 27
- logcrit
 - nsCamera.boards.LLNL_v1.IInl_v1, 169
 - nsCamera.boards.LLNL_v4.IInl_v4, 190
 - nsCamera.CameraAssembler.CameraAssembler, 91
 - nsCamera.comms.GigE.GigE, 134
 - nsCamera.comms.RS422.RS422, 209
 - nsCamera.sensors.sensorBase.sensorBase, 227
 - nsCamera.utils.misc.fakeCA, 120
- logcritbase
 - nsCamera.CameraAssembler.CameraAssembler, 91
 - nsCamera.utils.misc.fakeCA, 120
- logdebug
 - nsCamera.boards.LLNL_v1.IInl_v1, 170
 - nsCamera.boards.LLNL_v4.IInl_v4, 190
 - nsCamera.CameraAssembler.CameraAssembler, 92
 - nsCamera.comms.GigE.GigE, 134
 - nsCamera.comms.RS422.RS422, 209
 - nsCamera.sensors.sensorBase.sensorBase, 227
 - nsCamera.utils.misc.fakeCA, 120
- logdebugbase
 - nsCamera.CameraAssembler.CameraAssembler, 92
 - nsCamera.utils.misc.fakeCA, 120
- logerr
 - nsCamera.boards.LLNL_v1.IInl_v1, 170
 - nsCamera.boards.LLNL_v4.IInl_v4, 190
 - nsCamera.CameraAssembler.CameraAssembler, 92
 - nsCamera.comms.GigE.GigE, 134
 - nsCamera.comms.RS422.RS422, 209
 - nsCamera.sensors.sensorBase.sensorBase, 227
 - nsCamera.utils.misc.fakeCA, 120
- logerrbase
 - nsCamera.CameraAssembler.CameraAssembler, 92
 - nsCamera.utils.misc.fakeCA, 121
- loginfo
 - nsCamera.boards.LLNL_v1.IInl_v1, 170
 - nsCamera.boards.LLNL_v4.IInl_v4, 190
 - nsCamera.CameraAssembler.CameraAssembler, 92
 - nsCamera.comms.GigE.GigE, 134
 - nsCamera.comms.RS422.RS422, 209
 - nsCamera.sensors.sensorBase.sensorBase, 228
 - nsCamera.utils.misc.fakeCA, 121
- loginfobase
 - nsCamera.CameraAssembler.CameraAssembler, 92
 - nsCamera.utils.misc.fakeCA, 121
- loglabel
 - nsCamera.sensors.daedalus.daedalus, 115
 - nsCamera.sensors.icarus.icarus, 142
 - nsCamera.sensors.icarus2.icarus2, 149
- logtag
 - nsCamera.CameraAssembler.CameraAssembler, 92
 - nsCamera.utils.misc.fakeCA, 121
- logwarn
 - nsCamera.boards.LLNL_v1.IInl_v1, 170
 - nsCamera.boards.LLNL_v4.IInl_v4, 190
 - nsCamera.CameraAssembler.CameraAssembler, 93
 - nsCamera.comms.GigE.GigE, 135
 - nsCamera.comms.RS422.RS422, 209
 - nsCamera.sensors.sensorBase.sensorBase, 228
 - nsCamera.utils.misc.fakeCA, 121
- logwarnbase
 - nsCamera.CameraAssembler.CameraAssembler, 93
 - nsCamera.utils.misc.fakeCA, 121
- Main.c
 - ZestETM1Close, 329
 - ZestETM1Init, 329
- makeLogLabels
 - nsCamera.utils.misc, 30
- max
 - nsCamera.utils.Subregister.SubRegister, 229
- max_value
 - nsCamera.utils.Subregister.SubRegister, 230
- maxframe
 - nsCamera.sensors.daedalus.daedalus, 116
 - nsCamera.sensors.icarus.icarus, 143
 - nsCamera.sensors.icarus2.icarus2, 150
- maxheight
 - nsCamera.sensors.daedalus.daedalus, 116
 - nsCamera.sensors.icarus.icarus, 143
 - nsCamera.sensors.icarus2.icarus2, 150
- maxV
 - nsCamera.utils.Subregister.SubRegister, 230
- maxwidth
 - nsCamera.sensors.daedalus.daedalus, 116
 - nsCamera.sensors.icarus.icarus, 143
 - nsCamera.sensors.icarus2.icarus2, 150
- min
 - nsCamera.utils.Subregister.SubRegister, 230
- minframe
 - nsCamera.sensors.daedalus.daedalus, 116
 - nsCamera.sensors.icarus.icarus, 143
 - nsCamera.sensors.icarus2.icarus2, 150
- minV
 - nsCamera.utils.Subregister.SubRegister, 230
- mmReadoff
 - nsCamera.CameraAssembler.CameraAssembler, 64
- mode
 - nsCamera.comms.GigE.GigE, 135
 - nsCamera.comms.RS422.RS422, 209
- name
 - nsCamera.utils.Subregister.SubRegister, 230
- nargs
 - nsCamera.utils.FlatField, 23
- nframes
 - nsCamera.sensors.daedalus.daedalus, 116

- nsCamera.sensors.icarus.icarus, 143
 - nsCamera.sensors.icarus2.icarus2, 150
- nsCamera, 9
 - __all__, 10
- nsCamera.boards, 10
 - __all__, 10
- nsCamera.boards.LLNL_v1, 11
- nsCamera.boards.LLNL_v1.llnl_v1, 152
 - __init__, 153
 - ADC5_bipolar, 169
 - ADC5_mult, 169
 - ca, 169
 - checkStatus, 155
 - checkStatus2, 155
 - clearStatus, 155
 - configADCs, 156
 - daedalus_monitor_controls, 169
 - daedalus_subreg_aliases, 169
 - disarm, 156
 - dumpStatus, 157
 - enableLED, 158
 - getPressure, 159
 - getTemp, 159
 - getTimer, 160
 - icarus_monitor_controls, 169
 - icarus_subreg_aliases, 169
 - initBoard, 160
 - initPots, 161
 - initSensor, 162
 - latchPots, 162
 - logcrit, 169
 - logdebug, 170
 - logerr, 170
 - loginfo, 170
 - logwarn, 170
 - readSRAM, 163
 - registers, 170
 - reportEdgeDetects, 163
 - reportStatus, 164
 - resetTimer, 164
 - rs422_baud, 170
 - rs422_cmd_wait, 170
 - setLED, 165
 - setPowerSave, 165
 - setPPER, 166
 - softReboot, 166
 - startCapture, 167
 - subregisters, 171
 - subreglist, 171
 - VREF, 171
 - waitForSRAM, 168
- nsCamera.boards.LLNL_v4, 11
- nsCamera.boards.LLNL_v4.llnl_v4, 171
 - __init__, 173
- ADC5_bipolar, 189
- ADC5_mult, 189
- ca, 189
- checkStatus, 175
- checkStatus2, 175
- clearStatus, 175
- configADCs, 176
- daedalus_monitor_controls, 189
- daedalus_subreg_aliases, 189
- defoff, 189
- defsens, 189
- disarm, 176
- dumpStatus, 177
- enableLED, 178
- getPressure, 179
- getTemp, 180
- getTimer, 180
- icarus_monitor_controls, 189
- icarus_subreg_aliases, 190
- initBoard, 181
- initPots, 181
- initSensor, 181
- latchPots, 182
- logcrit, 190
- logdebug, 190
- logerr, 190
- loginfo, 190
- logwarn, 190
- readSRAM, 183
- registers, 190
- reportEdgeDetects, 183
- reportStatus, 184
- resetTimer, 185
- rs422_baud, 191
- rs422_cmd_wait, 191
- setLED, 185
- setPowerSave, 185
- setPPER, 186
- softReboot, 186
- startCapture, 187
- subregisters, 191
- subreglist, 191
- VREF, 191
- waitForSRAM, 188
- nsCamera.CameraAssembler, 11
- nsCamera.CameraAssembler.CameraAssembler, 39
 - __init__, 43
 - abort, 89
 - abortReadoff, 46
 - arch, 89
 - arm, 47
 - armed, 89
 - batchAcquire, 47
 - board, 89

boardname, 89
bytes2str, 48
checkCRC, 49
checkRegSet, 49
checkSensorVoltStat, 50
checkStatus, 50
checkStatus2, 50
clearStatus, 50
closeDevice, 50
commname, 89
comms, 90
configADCs, 51
currtime, 90
cycle, 90
disarm, 51
dumpNumpy, 51
dumpRegisters, 52
dumpStatus, 53
dumpSubregisters, 53
enableLED, 53
flattenlist, 54
FPGAboardtype, 90
FPGAinterfaces, 90
FPGAinvalid, 90
FPGANum, 90
FPGArad, 91
FPGAAsensor, 91
FPGAVersion, 91
getBoardInfo, 54
getEnter, 55
getManualTiming, 55
getMonV, 55
getPot, 56
getPotV, 57
getPressure, 58
getRegister, 59
getSensTemp, 59
getSubregister, 60
getTemp, 61
getTimer, 61
getTiming, 61
initBoard, 61
initialize, 61
initPots, 63
initPowerCheck, 63
initSensor, 64
inittime, 91
iplist, 91
latchPots, 64
logcrit, 91
logcritbase, 91
logdebug, 92
logdebugbase, 92
logerr, 92
logerrbase, 92
loginfo, 92
loginfobase, 92
logtag, 92
logwarn, 93
logwarnbase, 93
mmReadoff, 64
oldtime, 93
packageroot, 93
padToFull, 93
parsedtime, 93
parseReadoff, 65
partition, 65
payloaderror, 93
platform, 94
plotFrames, 65
port, 94
powerCheck, 65
printBoardInfo, 66
PY3, 94
pyth1, 94
pyth2, 94
python, 94
read, 94
readFrames, 67
readImgs, 67
readoff, 67
readSerial, 68
readSRAM, 68
reboot, 68
reinitialize, 68
reportEdgeDetects, 69
reportStatus, 69
resetTimer, 69
resolveSubreg, 69
saveFrames, 70
saveHDF, 71
saveNumpys, 72
saveTiffs, 74
savetime, 95
selectOscillator, 74
sendCMD, 74
sensmanual, 95
sensor, 95
sensorname, 95
sensorSpecific, 74
sens timing, 95
setArbTiming, 74
setExtClk, 75
setFrames, 75
setHighFullWell, 76
setInterlacing, 76
setLED, 77
setManualShutters, 77

- setManualTiming, [77](#)
- setPhiDelay, [77](#)
- setPot, [77](#)
- setPotV, [79](#)
- setPowerSave, [82](#)
- setPPER, [82](#)
- setRegister, [82](#)
- setRows, [83](#)
- setSubregister, [85](#)
- setTiming, [86](#)
- setTriggerDelay, [86](#)
- setZeroDeadTime, [87](#)
- startCapture, [87](#)
- str2bytes, [87](#)
- str2npararray, [87](#)
- submitMessages, [87](#)
- timeout, [95](#)
- trigtime, [95](#)
- unstringed, [96](#)
- verblevel, [96](#)
- verbmap, [96](#)
- verbose, [96](#)
- version, [96](#)
- waited, [96](#)
- waitForSRAM, [88](#)
- writeSerial, [88](#)
- nsCamera.comms, [12](#)
 - __all__, [13](#)
- nsCamera.comms.GigE, [13](#)
- nsCamera.comms.GigE.GigE, [122](#)
 - __init__, [124](#)
 - _zest, [133](#)
 - arm, [125](#)
 - ca, [133](#)
 - CardInfo, [133](#)
 - CardInfoP, [133](#)
 - closecard, [134](#)
 - closeDevice, [126](#)
 - Connection, [134](#)
 - dport, [134](#)
 - getCardInfo, [126](#)
 - getCardIP, [127](#)
 - logcrit, [134](#)
 - logdebug, [134](#)
 - logerr, [134](#)
 - loginfo, [134](#)
 - logwarn, [135](#)
 - mode, [135](#)
 - openDevice, [127](#)
 - payloadsize, [135](#)
 - readFrames, [128](#)
 - readoff, [129](#)
 - readSerial, [130](#)
 - readTimeout, [135](#)
 - sendCMD, [131](#)
 - skipError, [135](#)
 - writeSerial, [132](#)
 - writeTimeout, [135](#)
 - ZCountCards, [135](#)
 - ZErrorDict, [136](#)
 - ZESTETM1_CARD_INFO, [136](#)
 - ZOpenConnection, [136](#)
 - ZReadData, [136](#)
 - ZWriteData, [136](#)
- nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO, [231](#)
 - _fields_, [232](#)
 - ubyte4, [232](#)
 - ubyte6, [232](#)
- nsCamera.comms.RS422, [13](#)
- nsCamera.comms.RS422.RS422, [198](#)
 - __init__, [199](#)
 - _ser, [208](#)
 - arm, [201](#)
 - baud, [208](#)
 - ca, [208](#)
 - closeDevice, [201](#)
 - datatimeout, [208](#)
 - logcrit, [209](#)
 - logdebug, [209](#)
 - logerr, [209](#)
 - loginfo, [209](#)
 - logwarn, [209](#)
 - mode, [209](#)
 - par, [209](#)
 - payloadsize, [210](#)
 - port, [210](#)
 - PY3, [210](#)
 - read_timeout, [210](#)
 - readFrames, [202](#)
 - readoff, [203](#)
 - readSerial, [204](#)
 - sendCMD, [205](#)
 - serialClose, [207](#)
 - skipError, [210](#)
 - stop, [210](#)
 - write_timeout, [210](#)
 - writeSerial, [207](#)
- nsCamera.sensors, [14](#)
 - __all__, [14](#)
- nsCamera.sensors.daedalus, [14](#)
- nsCamera.sensors.daedalus.daedalus, [97](#)
 - __init__, [99](#)
 - bytesperpixel, [113](#)
 - ca, [113](#)
 - columns, [114](#)
 - detect, [114](#)
 - firstframe, [114](#)

- firstrow, [114](#)
- fpganumID, [114](#)
- getManualTiming, [100](#)
- getSensTemp, [101](#)
- height, [114](#)
- HFW, [115](#)
- interlacing, [115](#)
- lastframe, [115](#)
- lastrow, [115](#)
- loglabel, [115](#)
- maxframe, [116](#)
- maxheight, [116](#)
- maxwidth, [116](#)
- minframe, [116](#)
- nframes, [116](#)
- padToFull, [116](#)
- parseReadoff, [102](#)
- reportStatusSensor, [103](#)
- selectOscillator, [104](#)
- sens_registers, [116](#)
- sens_subregisters, [117](#)
- sensfam, [117](#)
- sensorSpecific, [104](#)
- setExtClk, [106](#)
- setHighFullWell, [107](#)
- setInterlacing, [107](#)
- setManualShutters, [109](#)
- setManualTiming, [109](#)
- setPhiDelay, [110](#)
- setTriggerDelay, [111](#)
- setZeroDeadTime, [112](#)
- specwarn, [117](#)
- toffset, [117](#)
- tslope, [117](#)
- width, [117](#)
- ZDT, [117](#), [118](#)
- nsCamera.sensors.icarus, [15](#)
- nsCamera.sensors.icarus.icarus, [137](#)
 - __init__, [139](#)
 - bytesperpixel, [141](#)
 - ca, [141](#)
 - checkSensorVoltStat, [140](#)
 - columns, [141](#)
 - detect, [141](#)
 - firstframe, [141](#)
 - firstrow, [141](#)
 - fpganumID, [142](#)
 - height, [142](#)
 - icarustype, [142](#)
 - interlacing, [142](#)
 - lastframe, [142](#)
 - lastrow, [142](#)
 - loglabel, [142](#)
 - maxframe, [143](#)
 - maxheight, [143](#)
 - maxwidth, [143](#)
 - minframe, [143](#)
 - nframes, [143](#)
 - padToFull, [143](#)
 - sens_registers, [143](#)
 - sens_subregisters, [144](#)
 - sensfam, [144](#)
 - sensorSpecific, [140](#)
 - specwarn, [144](#)
 - width, [144](#)
- nsCamera.sensors.icarus2, [15](#)
- nsCamera.sensors.icarus2.icarus2, [144](#)
 - __init__, [146](#)
 - bytesperpixel, [148](#)
 - ca, [148](#)
 - columns, [148](#)
 - detect, [148](#)
 - firstframe, [148](#)
 - firstrow, [148](#)
 - fpganumID, [149](#)
 - height, [149](#)
 - icarustype, [149](#)
 - interlacing, [149](#)
 - lastframe, [149](#)
 - lastrow, [149](#)
 - loglabel, [149](#)
 - maxframe, [150](#)
 - maxheight, [150](#)
 - maxwidth, [150](#)
 - minframe, [150](#)
 - nframes, [150](#)
 - padToFull, [150](#)
 - sens_registers, [150](#)
 - sens_subregisters, [151](#)
 - sensfam, [151](#)
 - sensorSpecific, [147](#)
 - specwarn, [151](#)
 - width, [151](#)
- nsCamera.sensors.sensorBase, [16](#)
- nsCamera.sensors.sensorBase.sensorBase, [211](#)
 - __init__, [212](#)
 - ca, [227](#)
 - checkSensorVoltStat, [213](#)
 - fpganumID, [227](#)
 - getManualTiming, [213](#)
 - getSensorStatus, [214](#)
 - getSensTemp, [214](#)
 - getTiming, [214](#)
 - init_board_specific, [216](#)
 - logcrit, [227](#)
 - logdebug, [227](#)
 - logerr, [227](#)
 - loginfo, [228](#)

- logwarn, 228
- parseReadoff, 216
- reportStatusSensor, 217
- selectOscillator, 217
- sensfam, 228
- setArbTiming, 218
- setExtClk, 220
- setHighFullWell, 221
- setInterlacing, 221
- setManualShutters, 221
- setManualTiming, 222
- setPhiDelay, 223
- setTiming, 224
- setTriggerDelay, 226
- setZeroDeadTime, 226
- nsCamera.utils, 16
 - __all__, 17
- nsCamera.utils.crc16pure, 17
 - _crc16, 18
 - CRC16_XMODEM_TABLE, 18
 - crc16xmodem, 18
- nsCamera.utils.FlatField, 19
 - action, 22
 - args, 22
 - default, 22
 - dest, 22
 - directory, 23
 - framelist, 23
 - generateFF, 20
 - getFilenames, 20
 - getROIvector, 20
 - help, 23
 - nargs, 23
 - parser, 23
 - removeFF, 21
 - removeFFall, 21
 - tslopes, 22
- nsCamera.utils.misc, 23
 - bytes2str, 24
 - checkCRC, 25
 - flattenlist, 25
 - generateFrames, 26
 - getEnter, 27
 - loadDumpedData, 27
 - makeLogLabels, 30
 - partition, 31
 - plotFrames, 32
 - saveTiffs, 33
 - str2bytes, 35
 - str2nparray, 35
- nsCamera.utils.misc.fakeCA, 118
 - __init__, 119
 - boardname, 120
 - logcrit, 120
 - logcritbase, 120
 - logdebug, 120
 - logdebugbase, 120
 - logerr, 120
 - logerrbase, 121
 - loginfo, 121
 - loginfobase, 121
 - logtag, 121
 - logwarn, 121
 - logwarnbase, 121
 - padToFull, 121
 - partition, 120
 - sensor, 122
 - sensorname, 122
- nsCamera.utils.Packet, 36
- nsCamera.utils.Packet.Packet, 192
 - __init__, 193
 - addr, 196
 - calculateCRC, 193
 - checkCRC, 194
 - checkReadPacket, 194
 - checkResponsePacket, 194
 - checkResponseString, 195
 - cmd, 196
 - crc, 197
 - data, 197
 - payload, 197
 - payload_length, 197
 - pktStr, 196
 - preamble, 197
 - PY3, 197
 - seqID, 197
 - type, 198
- nsCamera.utils.Subregister, 37
- nsCamera.utils.Subregister.SubRegister, 228
 - __init__, 229
 - addr, 229
 - max, 229
 - max_value, 230
 - maxV, 230
 - min, 230
 - minV, 230
 - name, 230
 - register, 230
 - resolution, 230
 - start_bit, 231
 - value, 231
 - width, 231
 - writable, 231
- oldtime
 - nsCamera.CameraAssembler.CameraAssembler, 93
- openDevice
 - nsCamera.comms.GigE.GigE, 127

- packageroot
 - nsCamera.CameraAssembler.CameraAssembler, 93
- padToFull
 - nsCamera.CameraAssembler.CameraAssembler, 93
 - nsCamera.sensors.daedalus.daedalus, 116
 - nsCamera.sensors.icarus.icarus, 143
 - nsCamera.sensors.icarus2.icarus2, 150
 - nsCamera.utils.misc.fakeCA, 121
- par
 - nsCamera.comms.RS422.RS422, 209
- parsedtime
 - nsCamera.CameraAssembler.CameraAssembler, 93
- parser
 - nsCamera.utils.FlatField, 23
- parseReadoff
 - nsCamera.CameraAssembler.CameraAssembler, 65
 - nsCamera.sensors.daedalus.daedalus, 102
 - nsCamera.sensors.sensorBase.sensorBase, 216
- partition
 - nsCamera.CameraAssembler.CameraAssembler, 65
 - nsCamera.utils.misc, 31
 - nsCamera.utils.misc.fakeCA, 120
- payload
 - nsCamera.utils.Packet.Packet, 197
- payload_length
 - nsCamera.utils.Packet.Packet, 197
- payloaderror
 - nsCamera.CameraAssembler.CameraAssembler, 93
- payloadsize
 - nsCamera.comms.GigE.GigE, 135
 - nsCamera.comms.RS422.RS422, 210
- pktStr
 - nsCamera.utils.Packet.Packet, 196
- platform
 - nsCamera.CameraAssembler.CameraAssembler, 94
- plotFrames
 - nsCamera.CameraAssembler.CameraAssembler, 65
 - nsCamera.utils.misc, 32
- port
 - nsCamera.CameraAssembler.CameraAssembler, 94
 - nsCamera.comms.RS422.RS422, 210
- powerCheck
 - nsCamera.CameraAssembler.CameraAssembler, 65
- preamble
 - nsCamera.utils.Packet.Packet, 197
- printBoardInfo
 - nsCamera.CameraAssembler.CameraAssembler, 66
- Private.h
 - closesocket, 333
 - SD_BOTH, 333
 - SOCKET, 335
 - ZestETM1_CloseConnection, 336
 - ZESTETM1_CONNECTION_HANDLE_MAGIC, 333
 - ZESTETM1_DEFAULT_TIMEOUT, 333
 - ZestETM1_EraseFlashSector, 336
 - ZESTETM1_ERROR, 333
 - ZESTETM1_ERROR_CONN, 333
 - ZESTETM1_ERROR_GENERAL, 334
 - ZESTETM1_ERROR_STRING, 334
 - ZestETM1_ErrorHandler, 340
 - ZestETM1_ErrorStrings, 340
 - ZestETM1_OpenConnection, 336
 - ZESTETM1_RATE_10MHz, 334
 - ZESTETM1_RATE_20MHz, 335
 - ZESTETM1_RATE_40MHz, 335
 - ZESTETM1_REVERSE, 335
 - ZestETM1_SendCommand, 338
 - ZestETM1_SPIReadWrite, 338
 - ZESTETM1_USER_DEVICE_ID, 335
 - ZestETM1_WriteFlash, 339
- PY3
 - nsCamera.CameraAssembler.CameraAssembler, 94
 - nsCamera.comms.RS422.RS422, 210
 - nsCamera.utils.Packet.Packet, 197
- pyth1
 - nsCamera.CameraAssembler.CameraAssembler, 94
- pyth2
 - nsCamera.CameraAssembler.CameraAssembler, 94
- python
 - nsCamera.CameraAssembler.CameraAssembler, 94
- read
 - nsCamera.CameraAssembler.CameraAssembler, 94
- read_timeout
 - nsCamera.comms.RS422.RS422, 210
- readFrames
 - nsCamera.CameraAssembler.CameraAssembler, 67
 - nsCamera.comms.GigE.GigE, 128
 - nsCamera.comms.RS422.RS422, 202
- readImgs
 - nsCamera.CameraAssembler.CameraAssembler, 67
- readoff
 - nsCamera.CameraAssembler.CameraAssembler, 67
 - nsCamera.comms.GigE.GigE, 129
 - nsCamera.comms.RS422.RS422, 203
- readSerial
 - nsCamera.CameraAssembler.CameraAssembler, 68
 - nsCamera.comms.GigE.GigE, 130
 - nsCamera.comms.RS422.RS422, 204
- readSRAM
 - nsCamera.boards.LLNL_v1.llnl_v1, 163
 - nsCamera.boards.LLNL_v4.llnl_v4, 183
 - nsCamera.CameraAssembler.CameraAssembler, 68
- readTimeout
 - nsCamera.comms.GigE.GigE, 135
- reboot
 - nsCamera.CameraAssembler.CameraAssembler, 68
- register

- nsCamera.utils.Subregister.SubRegister, 230
- registers
 - nsCamera.boards.LLNL_v1.llnl_v1, 170
 - nsCamera.boards.LLNL_v4.llnl_v4, 190
- reinitialize
 - nsCamera.CameraAssembler.CameraAssembler, 68
- removeFF
 - nsCamera.utils.FlatField, 21
- removeFFall
 - nsCamera.utils.FlatField, 21
- reportEdgeDetects
 - nsCamera.boards.LLNL_v1.llnl_v1, 163
 - nsCamera.boards.LLNL_v4.llnl_v4, 183
 - nsCamera.CameraAssembler.CameraAssembler, 69
- reportStatus
 - nsCamera.boards.LLNL_v1.llnl_v1, 164
 - nsCamera.boards.LLNL_v4.llnl_v4, 184
 - nsCamera.CameraAssembler.CameraAssembler, 69
- reportStatusSensor
 - nsCamera.sensors.daedalus.daedalus, 103
 - nsCamera.sensors.sensorBase.sensorBase, 217
- resetTimer
 - nsCamera.boards.LLNL_v1.llnl_v1, 164
 - nsCamera.boards.LLNL_v4.llnl_v4, 185
 - nsCamera.CameraAssembler.CameraAssembler, 69
- resolution
 - nsCamera.utils.Subregister.SubRegister, 230
- resolveSubreg
 - nsCamera.CameraAssembler.CameraAssembler, 69
- rs422_baud
 - nsCamera.boards.LLNL_v1.llnl_v1, 170
 - nsCamera.boards.LLNL_v4.llnl_v4, 191
- rs422_cmd_wait
 - nsCamera.boards.LLNL_v1.llnl_v1, 170
 - nsCamera.boards.LLNL_v4.llnl_v4, 191
- saveFrames
 - nsCamera.CameraAssembler.CameraAssembler, 70
- saveHDF
 - nsCamera.CameraAssembler.CameraAssembler, 71
- saveNumpys
 - nsCamera.CameraAssembler.CameraAssembler, 72
- saveTiffs
 - nsCamera.CameraAssembler.CameraAssembler, 74
 - nsCamera.utils.misc, 33
- savetime
 - nsCamera.CameraAssembler.CameraAssembler, 95
- SD_BOTH
 - Private.h, 333
- selectOscillator
 - nsCamera.CameraAssembler.CameraAssembler, 74
 - nsCamera.sensors.daedalus.daedalus, 104
 - nsCamera.sensors.sensorBase.sensorBase, 217
- sendCMD
 - nsCamera.CameraAssembler.CameraAssembler, 74
 - nsCamera.comms.GigE.GigE, 131
 - nsCamera.comms.RS422.RS422, 205
- sens_registers
 - nsCamera.sensors.daedalus.daedalus, 116
 - nsCamera.sensors.icarus.icarus, 143
 - nsCamera.sensors.icarus2.icarus2, 150
- sens_subregisters
 - nsCamera.sensors.daedalus.daedalus, 117
 - nsCamera.sensors.icarus.icarus, 144
 - nsCamera.sensors.icarus2.icarus2, 151
- sensfam
 - nsCamera.sensors.daedalus.daedalus, 117
 - nsCamera.sensors.icarus.icarus, 144
 - nsCamera.sensors.icarus2.icarus2, 151
 - nsCamera.sensors.sensorBase.sensorBase, 228
- sensmanual
 - nsCamera.CameraAssembler.CameraAssembler, 95
- sensor
 - nsCamera.CameraAssembler.CameraAssembler, 95
 - nsCamera.utils.misc.fakeCA, 122
- sensorname
 - nsCamera.CameraAssembler.CameraAssembler, 95
 - nsCamera.utils.misc.fakeCA, 122
- sensorSpecific
 - nsCamera.CameraAssembler.CameraAssembler, 74
 - nsCamera.sensors.daedalus.daedalus, 104
 - nsCamera.sensors.icarus.icarus, 140
 - nsCamera.sensors.icarus2.icarus2, 147
- senstiming
 - nsCamera.CameraAssembler.CameraAssembler, 95
- seqID
 - nsCamera.utils.Packet.Packet, 197
- serialClose
 - nsCamera.comms.RS422.RS422, 207
- setArbTiming
 - nsCamera.CameraAssembler.CameraAssembler, 74
 - nsCamera.sensors.sensorBase.sensorBase, 218
- setExtClk
 - nsCamera.CameraAssembler.CameraAssembler, 75
 - nsCamera.sensors.daedalus.daedalus, 106
 - nsCamera.sensors.sensorBase.sensorBase, 220
- setFrames
 - nsCamera.CameraAssembler.CameraAssembler, 75
- setHighFullWell
 - nsCamera.CameraAssembler.CameraAssembler, 76
 - nsCamera.sensors.daedalus.daedalus, 107
 - nsCamera.sensors.sensorBase.sensorBase, 221
- setInterlacing
 - nsCamera.CameraAssembler.CameraAssembler, 76
 - nsCamera.sensors.daedalus.daedalus, 107
 - nsCamera.sensors.sensorBase.sensorBase, 221
- setLED
 - nsCamera.boards.LLNL_v1.llnl_v1, 165

- nsCamera.boards.LLNL_v4.IInl_v4, [185](#)
- nsCamera.CameraAssembler.CameraAssembler, [77](#)
- setManualShutters
 - nsCamera.CameraAssembler.CameraAssembler, [77](#)
 - nsCamera.sensors.daedalus.daedalus, [109](#)
 - nsCamera.sensors.sensorBase.sensorBase, [221](#)
- setManualTiming
 - nsCamera.CameraAssembler.CameraAssembler, [77](#)
 - nsCamera.sensors.daedalus.daedalus, [109](#)
 - nsCamera.sensors.sensorBase.sensorBase, [222](#)
- setPhiDelay
 - nsCamera.CameraAssembler.CameraAssembler, [77](#)
 - nsCamera.sensors.daedalus.daedalus, [110](#)
 - nsCamera.sensors.sensorBase.sensorBase, [223](#)
- setPot
 - nsCamera.CameraAssembler.CameraAssembler, [77](#)
- setPotV
 - nsCamera.CameraAssembler.CameraAssembler, [79](#)
- setPowerSave
 - nsCamera.boards.LLNL_v1.IInl_v1, [165](#)
 - nsCamera.boards.LLNL_v4.IInl_v4, [185](#)
 - nsCamera.CameraAssembler.CameraAssembler, [82](#)
- setPPER
 - nsCamera.boards.LLNL_v1.IInl_v1, [166](#)
 - nsCamera.boards.LLNL_v4.IInl_v4, [186](#)
 - nsCamera.CameraAssembler.CameraAssembler, [82](#)
- setRegister
 - nsCamera.CameraAssembler.CameraAssembler, [82](#)
- setRows
 - nsCamera.CameraAssembler.CameraAssembler, [83](#)
- setSubregister
 - nsCamera.CameraAssembler.CameraAssembler, [85](#)
- setTiming
 - nsCamera.CameraAssembler.CameraAssembler, [86](#)
 - nsCamera.sensors.sensorBase.sensorBase, [224](#)
- setTriggerDelay
 - nsCamera.CameraAssembler.CameraAssembler, [86](#)
 - nsCamera.sensors.daedalus.daedalus, [111](#)
 - nsCamera.sensors.sensorBase.sensorBase, [226](#)
- setZeroDeadTime
 - nsCamera.CameraAssembler.CameraAssembler, [87](#)
 - nsCamera.sensors.daedalus.daedalus, [112](#)
 - nsCamera.sensors.sensorBase.sensorBase, [226](#)
- skipError
 - nsCamera.comms.GigE.GigE, [135](#)
 - nsCamera.comms.RS422.RS422, [210](#)
- SOCKET
 - Private.h, [335](#)
- softReboot
 - nsCamera.boards.LLNL_v1.IInl_v1, [166](#)
 - nsCamera.boards.LLNL_v4.IInl_v4, [186](#)
- specwarn
 - nsCamera.sensors.daedalus.daedalus, [117](#)
 - nsCamera.sensors.icarus.icarus, [144](#)
- nsCamera.sensors.icarus2.icarus2, [151](#)
- start_bit
 - nsCamera.utils.Subregister.SubRegister, [231](#)
- startCapture
 - nsCamera.boards.LLNL_v1.IInl_v1, [167](#)
 - nsCamera.boards.LLNL_v4.IInl_v4, [187](#)
 - nsCamera.CameraAssembler.CameraAssembler, [87](#)
- stop
 - nsCamera.comms.RS422.RS422, [210](#)
- str2bytes
 - nsCamera.CameraAssembler.CameraAssembler, [87](#)
 - nsCamera.utils.misc, [35](#)
- str2narray
 - nsCamera.CameraAssembler.CameraAssembler, [87](#)
 - nsCamera.utils.misc, [35](#)
- submitMessages
 - nsCamera.CameraAssembler.CameraAssembler, [87](#)
- subregisters
 - nsCamera.boards.LLNL_v1.IInl_v1, [171](#)
 - nsCamera.boards.LLNL_v4.IInl_v4, [191](#)
- subreglist
 - nsCamera.boards.LLNL_v1.IInl_v1, [171](#)
 - nsCamera.boards.LLNL_v4.IInl_v4, [191](#)
- timeout
 - nsCamera.CameraAssembler.CameraAssembler, [95](#)
- toffset
 - nsCamera.sensors.daedalus.daedalus, [117](#)
- trigtime
 - nsCamera.CameraAssembler.CameraAssembler, [95](#)
- tslope
 - nsCamera.sensors.daedalus.daedalus, [117](#)
- tslopes
 - nsCamera.utils.FlatField, [22](#)
- type
 - nsCamera.utils.Packet.Packet, [198](#)
- ubyte4
 - nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO, [232](#)
- ubyte6
 - nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO, [232](#)
- unstringed
 - nsCamera.CameraAssembler.CameraAssembler, [96](#)
- UPnP.c
 - _CRT_SECURE_NO_WARNINGS, [344](#)
 - _strnicmp, [344](#)
 - ZESTETM1_COMMAND_GET_SETTINGS, [344](#)
 - ZestETM1_GetAllAdapters, [345](#)
 - ZestETM1_GetCardInfo, [346](#)
 - ZestETM1_HTTPGet, [347](#)
 - ZestETM1_JoinGroup, [349](#)
 - ZestETM1_LeaveGroup, [349](#)
 - ZestETM1_ReadSettings, [350](#)

- ZestETM1_SearchReq, 353
- ZESTETM1_UPNP_ADDR, 344
- ZESTETM1_UPNP_PORT, 344
- ZestETM1CountCards, 350
- ZestETM1FreeCards, 352
- ZestETM1GetCardInfo, 353
- value
 - nsCamera.utils.Subregister.SubRegister, 231
- verblevel
 - nsCamera.CameraAssembler.CameraAssembler, 96
- verbmap
 - nsCamera.CameraAssembler.CameraAssembler, 96
- verbose
 - nsCamera.CameraAssembler.CameraAssembler, 96
- version
 - nsCamera.CameraAssembler.CameraAssembler, 96
- VREF
 - nsCamera.boards.LLNL_v1.IInl_v1, 171
 - nsCamera.boards.LLNL_v4.IInl_v4, 191
- waited
 - nsCamera.CameraAssembler.CameraAssembler, 96
- waitForSRAM
 - nsCamera.boards.LLNL_v1.IInl_v1, 168
 - nsCamera.boards.LLNL_v4.IInl_v4, 188
 - nsCamera.CameraAssembler.CameraAssembler, 88
- width
 - nsCamera.sensors.daedalus.daedalus, 117
 - nsCamera.sensors.icarus.icarus, 144
 - nsCamera.sensors.icarus2.icarus2, 151
 - nsCamera.utils.Subregister.SubRegister, 231
- writable
 - nsCamera.utils.Subregister.SubRegister, 231
- write_timeout
 - nsCamera.comms.RS422.RS422, 210
- writeSerial
 - nsCamera.CameraAssembler.CameraAssembler, 88
 - nsCamera.comms.GigE.GigE, 132
 - nsCamera.comms.RS422.RS422, 207
- writeTimeout
 - nsCamera.comms.GigE.GigE, 135
- ZCountCards
 - nsCamera.comms.GigE.GigE, 135
- ZDT
 - nsCamera.sensors.daedalus.daedalus, 117, 118
- ZErrorDict
 - nsCamera.comms.GigE.GigE, 136
- ZestETM1.h
 - ZESTETM1_CONNECTION, 365
 - ZESTETM1_CONNECTION_TYPE, 365
 - ZESTETM1_ERROR_BASE, 364
 - ZESTETM1_ERROR_FUNC, 365
 - ZESTETM1_HANDLE, 365
 - ZESTETM1_ILLEGAL_CONNECTION, 366
 - ZESTETM1_ILLEGAL_PARAMETER, 366
 - ZESTETM1_ILLEGAL_STATUS_CODE, 366
 - ZESTETM1_INFO_BASE, 364
 - ZESTETM1_INTERNAL_ERROR, 366
 - ZESTETM1_INVALID_CONNECTION_TYPE, 366
 - ZESTETM1_MAX_ERROR, 366
 - ZESTETM1_MAX_INFO, 366
 - ZESTETM1_MAX_WARNING, 366
 - ZESTETM1_NULL_PARAMETER, 366
 - ZESTETM1_OUT_OF_MEMORY, 366
 - ZESTETM1_SOCKET_CLOSED, 366
 - ZESTETM1_SOCKET_ERROR, 366
 - ZESTETM1_SPI_RATE, 365
 - ZESTETM1_SPI_RATE_17_5, 366
 - ZESTETM1_SPI_RATE_35, 365
 - ZESTETM1_SPI_RATE_8_75, 366
 - ZESTETM1_STATUS, 366
 - ZESTETM1_SUCCESS, 366
 - ZESTETM1_TIMEOUT, 366
 - ZESTETM1_TYPE_TCP, 365
 - ZESTETM1_TYPE_UDP, 365
 - ZESTETM1_VERSION_FALLBACK, 364
 - ZESTETM1_WARNING_BASE, 364
 - ZestETM1Close, 367
 - ZestETM1CloseConnection, 367
 - ZestETM1CountCards, 367
 - ZestETM1FreeCards, 369
 - ZestETM1GetCardInfo, 369
 - ZestETM1GetErrorMessage, 370
 - ZestETM1Init, 370
 - ZestETM1OpenConnection, 370
 - ZestETM1ReadData, 371
 - ZestETM1ReadRegister, 371
 - ZestETM1RegisterErrorHandler, 372
 - ZestETM1SetInterrupt, 372
 - ZestETM1SPIReadWrite, 373
 - ZestETM1WriteData, 373
 - ZestETM1WriteRegister, 374
 - ZESTETM1_CARD_INFO, 364
 - nsCamera.comms.GigE.GigE, 136
 - ZestETM1_CloseConnection
 - Data.c, 305
 - Private.h, 336
 - ZESTETM1_COMMAND_GET_SETTINGS
 - UPnP.c, 344
 - ZESTETM1_COMMAND_MAILBOX_INT
 - Data.c, 304
 - ZESTETM1_COMMAND_READ_REG
 - Data.c, 304
 - ZESTETM1_COMMAND_SPI
 - Data.c, 304
 - ZESTETM1_COMMAND_WRITE_REG
 - Data.c, 304

ZESTETM1_CONNECTION
 ZestETM1.h, [365](#)
ZESTETM1_CONNECTION_HANDLE_MAGIC
 Private.h, [333](#)
ZESTETM1_CONNECTION_STRUCT, [332](#)
ZESTETM1_CONNECTION_TYPE
 ZestETM1.h, [365](#)
ZESTETM1_DEFAULT_TIMEOUT
 Private.h, [333](#)
ZestETM1_EraseFlashSector
 Private.h, [336](#)
ZESTETM1_ERROR
 Private.h, [333](#)
ZESTETM1_ERROR_BASE
 ZestETM1.h, [364](#)
ZESTETM1_ERROR_CONN
 Private.h, [333](#)
ZESTETM1_ERROR_FUNC
 ZestETM1.h, [365](#)
ZESTETM1_ERROR_GENERAL
 Private.h, [334](#)
ZESTETM1_ERROR_STRING
 Private.h, [334](#)
ZestETM1_ErrorHandler
 Error.c, [327](#)
 Private.h, [340](#)
ZestETM1_ErrorStrings
 Error.c, [327](#)
 Private.h, [340](#)
ZESTETM1_GET_SETTINGS_CMD, [343](#)
ZESTETM1_GET_SETTINGS_RESPONSE, [343](#)
ZestETM1_GetAllAdapters
 UPnP.c, [345](#)
ZestETM1_GetCardInfo
 UPnP.c, [346](#)
ZESTETM1_HANDLE
 ZestETM1.h, [365](#)
ZestETM1_HTTPGet
 UPnP.c, [347](#)
ZESTETM1_ILLEGAL_CONNECTION
 ZestETM1.h, [366](#)
ZESTETM1_ILLEGAL_PARAMETER
 ZestETM1.h, [366](#)
ZESTETM1_ILLEGAL_STATUS_CODE
 ZestETM1.h, [366](#)
ZESTETM1_INFO_BASE
 ZestETM1.h, [364](#)
ZESTETM1_INTERNAL_ERROR
 ZestETM1.h, [366](#)
ZESTETM1_INVALID_CONNECTION_TYPE
 ZestETM1.h, [366](#)
ZestETM1_JoinGroup
 UPnP.c, [349](#)
ZestETM1_LeaveGroup
 UPnP.c, [349](#)
ZESTETM1_MAILBOX_INT_CMD, [303](#)
ZESTETM1_MAILBOX_INT_RESPONSE, [303](#)
ZESTETM1_MAX_ERROR
 ZestETM1.h, [366](#)
ZESTETM1_MAX_INFO
 ZestETM1.h, [366](#)
ZESTETM1_MAX_WARNING
 ZestETM1.h, [366](#)
ZESTETM1_NULL_PARAMETER
 ZestETM1.h, [366](#)
ZestETM1_OpenConnection
 Data.c, [305](#)
 Private.h, [336](#)
ZESTETM1_OUT_OF_MEMORY
 ZestETM1.h, [366](#)
ZESTETM1_RATE_10MHz
 Private.h, [334](#)
ZESTETM1_RATE_20MHz
 Private.h, [335](#)
ZESTETM1_RATE_40MHz
 Private.h, [335](#)
ZESTETM1_READ_REG_CMD, [303](#)
ZESTETM1_READ_REG_RESPONSE, [303](#)
ZestETM1_ReadData
 Data.c, [307](#)
ZestETM1_ReadSettings
 UPnP.c, [350](#)
ZESTETM1_REVERSE
 Private.h, [335](#)
ZestETM1_SearchReq
 UPnP.c, [353](#)
ZestETM1_SendCommand
 Data.c, [308](#)
 Private.h, [338](#)
ZESTETM1_SOCKET_CLOSED
 ZestETM1.h, [366](#)
ZESTETM1_SOCKET_ERROR
 ZestETM1.h, [366](#)
ZESTETM1_SPI_RATE
 ZestETM1.h, [365](#)
ZESTETM1_SPI_RATE_17_5
 ZestETM1.h, [366](#)
ZESTETM1_SPI_RATE_35
 ZestETM1.h, [365](#)
ZESTETM1_SPI_RATE_8_75
 ZestETM1.h, [366](#)
ZestETM1_SPIReadWrite
 Data.c, [308](#)
 Private.h, [338](#)
ZESTETM1_STATUS
 ZestETM1.h, [366](#)
ZESTETM1_SUCCESS
 ZestETM1.h, [366](#)

ZESTETM1_TIMEOUT
 ZestETM1.h, [366](#)

ZESTETM1_TYPE_TCP
 ZestETM1.h, [365](#)

ZESTETM1_TYPE_UDP
 ZestETM1.h, [365](#)

ZESTETM1_UPNP_ADDR
 UPnP.c, [344](#)

ZESTETM1_UPNP_PORT
 UPnP.c, [344](#)

ZESTETM1_USER_DEVICE_ID
 Private.h, [335](#)

ZESTETM1_VERSION_FALLBACK
 ZestETM1.h, [364](#)

ZESTETM1_WARNING_BASE
 ZestETM1.h, [364](#)

ZESTETM1_WRITE_REG_CMD, [302](#)

ZESTETM1_WRITE_REG_RESPONSE, [302](#)

ZestETM1_WriteData
 Data.c, [309](#)

ZestETM1_WriteFlash
 Private.h, [339](#)

ZestETM1Close
 Main.c, [329](#)
 ZestETM1.h, [367](#)

ZestETM1CloseConnection
 Data.c, [311](#)
 ZestETM1.h, [367](#)

ZestETM1CountCards
 UPnP.c, [350](#)
 ZestETM1.h, [367](#)

ZestETM1FreeCards
 UPnP.c, [352](#)
 ZestETM1.h, [369](#)

ZestETM1GetCardInfo
 UPnP.c, [353](#)
 ZestETM1.h, [369](#)

ZestETM1GetErrorMessage
 Error.c, [326](#)
 ZestETM1.h, [370](#)

ZestETM1Init
 Main.c, [329](#)
 ZestETM1.h, [370](#)

ZestETM1OpenConnection
 Data.c, [311](#)
 ZestETM1.h, [370](#)

ZestETM1ReadData
 Data.c, [311](#)
 ZestETM1.h, [371](#)

ZestETM1ReadRegister
 Data.c, [312](#)
 ZestETM1.h, [371](#)

ZestETM1RegisterErrorHandler
 Error.c, [326](#)
 ZestETM1.h, [372](#)

ZestETM1SetInterrupt
 Data.c, [313](#)
 ZestETM1.h, [372](#)

ZestETM1SPIReadWrite
 Data.c, [313](#)
 ZestETM1.h, [373](#)

ZestETM1WriteData
 Data.c, [314](#)
 ZestETM1.h, [373](#)

ZestETM1WriteRegister
 Data.c, [314](#)
 ZestETM1.h, [374](#)

ZOpenConnection
 nsCamera.comms.GigE.GigE, [136](#)

ZReadData
 nsCamera.comms.GigE.GigE, [136](#)

ZWriteData
 nsCamera.comms.GigE.GigE, [136](#)