

# The CDLidarCaptureMethod Class

Scott D. Brown  
Digital Imaging and Remote Sensing Laboratory  
Rochester Institute of Technology  
Rochester, NY 14623

March 9, 2006

## Contents

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Copyright Notice</b>                               | <b>1</b> |
| <b>2</b> | <b>Overview</b>                                       | <b>1</b> |
| 2.1      | XML Configuration . . . . .                           | 1        |
| 2.2      | Document Version . . . . .                            | 2        |
| <b>3</b> | <b>The CDLidarCaptureMethod class</b>                 | <b>2</b> |
| 3.1      | Class Dependancies . . . . .                          | 2        |
| 3.2      | Class Declaration . . . . .                           | 2        |
| 3.3      | Private Type Definitions . . . . .                    | 3        |
| 3.3.1    | The task data header . . . . .                        | 3        |
| 3.3.2    | The pulse data header . . . . .                       | 4        |
| 3.4      | Private Variable Declarations . . . . .               | 4        |
| 3.4.1    | The scan index counter . . . . .                      | 4        |
| 3.4.2    | The pixel buffer . . . . .                            | 4        |
| 3.4.3    | Generate output files per pulse flag . . . . .        | 5        |
| 3.4.4    | Poisson photon arrival statistics variables . . . . . | 5        |
| 3.5      | Constructors . . . . .                                | 5        |
| 3.5.1    | Default Constructor . . . . .                         | 5        |
| 3.6      | Destructor . . . . .                                  | 5        |
| 3.7      | Virtual Functions . . . . .                           | 6        |
| 3.7.1    | The initialization function . . . . .                 | 6        |

|          |   |           |
|----------|---|-----------|
| 3.7.2    | The task start function . . . . .               | 6         |
| 3.7.3    | The task end function . . . . .                 | 8         |
| 3.7.4    | The capture start function . . . . .            | 8         |
| 3.7.5    | The capture end function . . . . .              | 9         |
| 3.7.6    | The solution computation function . . . . .     | 10        |
| 3.7.7    | Open the output image file . . . . .            | 10        |
| 3.7.8    | Write the task data to the file . . . . .       | 13        |
| 3.7.9    | Write the pulse data header to a file . . . . . | 15        |
| 3.7.10   | The primary function . . . . .                  | 17        |
| 3.8      | XML Support Functions . . . . .                 | 19        |
| 3.8.1    | Load from an XML DOM document . . . . .         | 19        |
| 3.8.2    | Store into an XML DOM document . . . . .        | 21        |
| <b>4</b> | <b>Header file</b>                              | <b>22</b> |
| <b>5</b> | <b>Source file</b>                              | <b>23</b> |
| <b>6</b> | <b>Revision History</b>                         | <b>24</b> |

## 1 Copyright Notice

Copyright Rochester Institute of Technology, Chester F. Carlson Center for Imaging Science, Digital Imaging and Remote Sensing (DIRS) Laboratory

The software source code contained in this document is the intellectual property of Rochester Institute of Technology (RIT) and is considered confidential information.

## 2 Overview

This class is a capture method (derived from `CDCaptureMethod`) that was written to handle (primarily) the output of a traditional framing array or single detector LIDAR system.

## 2.1 XML Configuration

This capture method is specified by supplying `lidar` as the `name` attribute in the `capturemethod` element. The only configuration option for this capture method at this time is the compression flag.

```
1  <XML Configuration 1>≡
    <capturemethod name="lidar">
      <outputfiles perpulse="1"></outputfiles>
      <poissonstatistics enabled="1"></poissonstatistics>
    </capturemethod>
```

The `outputfiles` element controls if the model will create individual files for each pulse. The `poissonstatistics` element controls if the model will impose Poisson arrival statistics to the photon counts.

## 2.2 Document Version

The following chunk stores the current document version. This is accessible through the class namespace so that the programmer can set up query mechanisms from the application to track the versions used in a specific build.

```
2a  <Document version 2a>≡ (23)
    const char * CDLidarCaptureMethod::version = "$Revision: 1.43 $";
```

### 3 The CDLidarCaptureMethod class

#### 3.1 Class Dependancies

This class uses the following classes as private and public :

- The CDBand class is used to store the “bands” defined for the simulation.
- The CDSolution class is used to store the spatially and spectrally over-sampled radiances reaching the detector array.

#### 3.2 Class Declaration

```

2b  <CDLidarCaptureMethod Class 2b>≡ (22)
    class CDLidarCaptureMethod : public CDCaptureMethod
    {
    public:
        <CDLidarCaptureMethod Constructor Declarations 5c>
        <CDLidarCaptureMethod Destructor Declaration 5e>
        <CDLidarCaptureMethod Public Virtual Function Declarations 6b>

    private:
        <CDLidarCaptureMethod Private Type Definitions 3>
        <CDLidarCaptureMethod Private Variable Declarations 4b>

    public:
        static const char * version;

    };

```

### 3.3 Private Type Definitions

#### 3.3.1 The task data header

This structure is used to store all the information that is written to the output file at the start of a new task. It contains information about the instrument, the laser source, the time gate for the pulse data, and the number of pulses in the task. The total size is 140 bytes.

```

3  <CDLidarCaptureMethod Private Type Definitions 3>≡ (2b) 4a▷
    typedef struct _SimulationDataHeader {
        char          idString[ 64 ];
        unsigned int  endianFlag;
        char          generationDate[ 32 ];
        double        focalLength;
        double        xPixelPitch;
        double        yPixelPitch;
        unsigned int  xPixelCount;
        unsigned int  yPixelCount;
        double        pulseRate;
        double        pulseDuration;
        double        pulsePower;
        double        spectralPeak;
        double        spectralWidth;
        double        timeGateStart;
        double        timeGateStop;
        double        timeGateDelta;
        unsigned int  timeBinCount;
        unsigned int  pulseCount;
    } TaskDataHeader;

```

Defines:

`TaskDataHeader`, used in chunk 14.

### 3.3.2 The pulse data header

This structure is used to store all the information that is written out at the start of each pulse. It contains the time that the pulse left, the platform location and orientation and any platform relative along and across track pointing of the instrument (e.g. scanning). The size is 72 bytes.

```
4a <CDLidarCaptureMethod Private Type Definitions 3>+≡ (2b) <3
    typedef struct _PulseDataHeader {
        double          pulseTime;
        double          xPlatformPosition;
        double          yPlatformPosition;
        double          zPlatformPosition;
        double          xPlatformAngle;
        double          yPlatformAngle;
        double          zPlatformAngle;
        double          alongTrackAngle;
        double          acrossTrackAngle;
    } PulseDataHeader;
```

Defines:

`PulseDataHeader`, used in chunk 16.

## 3.4 Private Variable Declarations

### 3.4.1 The scan index counter

The `pulseIndex` variable tracks the current scan/capture being rendered within a given task (see the `CDTask` class). This variable is used only to create useful filenames when we create a different filename for each scan/capture.

```
4b <CDLidarCaptureMethod Private Variable Declarations 4b>≡ (2b) 4c>
    int pulseIndex;
```

Defines:

`pulseIndex`, used in chunks 5d, 7, and 11.

### 3.4.2 The pixel buffer

The `pixelBuffer` is a storage buffer that is sized to hold the temporal returns for a single pixel. That way we can buffer it and write it out all at once since `CDEnviImage` uses unbuffered I/O to avoid 64-bit file size issues.

```
4c <CDLidarCaptureMethod Private Variable Declarations 4b>+≡ (2b) <4b 5a>
    double * pixelBuffer;
```

Defines:

`pixelBuffer`, used in chunks 5d, 6c, and 18.

### 3.4.3 Generate output files per pulse flag

This variable stores if the object should create a separate file for each pulse.

5a  $\langle$ CDLidarCaptureMethod Private Variable Declarations 4b $\rangle$ + $\equiv$  (2b)  $\langle$ 4c 5b $\rangle$   
`bool outputPerPulse;`

Defines:

`outputPerPulse`, used in chunks 5d, 7, 9–11, 20, and 21b.

### 3.4.4 Poisson photon arrival statistics variables

The `enablePoissonStatistics` variables variable stores if this object should impose poisson photon arrival statistics. The `rng` variable is an instance of a random number generator used to compute values from the Poisson distribution.

5b  $\langle$ CDLidarCaptureMethod Private Variable Declarations 4b $\rangle$ + $\equiv$  (2b)  $\langle$ 5a $\rangle$   
`bool enablePoissonStatistics;`  
`CDRandomNumber rng;`

Defines:

`enablePoissonStatistics`, used in chunks 5d, 18, 20, and 21b.  
`rng`, used in chunks 5d and 18.

## 3.5 Constructors

### 3.5.1 Default Constructor

5c  $\langle$ CDLidarCaptureMethod Constructor Declarations 5c $\rangle$  $\equiv$  (2b)  
`CDLidarCaptureMethod( void );`

5d  $\langle$ CDLidarCaptureMethod Constructor Implementations 5d $\rangle$  $\equiv$  (23)  
`CDLidarCaptureMethod::CDLidarCaptureMethod( void )`  
`: CDCaptureMethod( "Lidar", "Time-Gated Lidar Capture" ),`  
`pulseIndex( 0 ), pixelBuffer( 0 ), outputPerPulse( false ),`  
`enablePoissonStatistics( false ), rng()`  
`{`  
`}`

Defines:

`CDLidarCaptureMethod::CDLidarCaptureMethod(string,string)`, never used.

Uses `enablePoissonStatistics` 5b, `outputPerPulse` 5a, `pixelBuffer` 4c, `pulseIndex` 4b, and `rng` 5b.

### 3.6 Destructor

5e  $\langle$ CDLidarCaptureMethod Destructor Declaration 5e $\rangle$  $\equiv$  (2b)  
`virtual ~CDLidarCaptureMethod( void );`

6a  $\langle$ CDLidarCaptureMethod Destructor Implementation 6a $\rangle \equiv$  (23)  
`CDLidarCaptureMethod::~CDLidarCaptureMethod( void )`  
`{`  
`}`

Defines:

`CDLidarCaptureMethod::~CDLidarCaptureMethod(void)`, never used.

## 3.7 Virtual Functions

### 3.7.1 The initialization function

For this capture method, there is really nothing to do at initialization time since this method does not perform any instrument specific operations such as spectral intergration, spatial integration, instrument specific noise.

6b  $\langle$ CDLidarCaptureMethod Public Virtual Function Declarations 6b $\rangle \equiv$  (2b) 6d $\triangleright$   
`bool init( CDSimulation * simulation, CDBand * band );`

6c  $\langle$ CDLidarCaptureMethod Public Virtual Function Implementations 6c $\rangle \equiv$  (23) 7 $\triangleright$   
`bool`  
`CDLidarCaptureMethod::init( CDSimulation * simulation, CDBand * band )`  
`{`  
`std::cout << "    Using generic LIDAR capture method" << std::endl;`  
`std::cout << "        Output image will have "`  
`<< band->getInstrument()->getSignalGate().getBinCount()`  
`<< " time steps" << std::endl << std::endl;`  
  
`// allocate the pixel buffer`  
`this->pixelBuffer =`  
`new double[ band->getInstrument()->getSignalGate().getBinCount() ];`  
  
`return true;`  
`}`

Defines:

`CDLidarCaptureMethod::init(CDBand*)`, never used.

Uses `pixelBuffer` 4c.

### 3.7.2 The task start function

This function is called at the start of a task. For this capture method, we write out a separate header file for this task if we are generating individual files for each scan/capture.

6d  $\langle$ CDLidarCaptureMethod Public Virtual Function Declarations 6b $\rangle + \equiv$  (2b)  $\langle$ 6b 8a $\rangle$   
`virtual bool startTask( CDSimulation * simulation, CDTask * task,`  
`CDBand * band, long captureCount, const CDDateTime & time );`

```

7  <CDLidarCaptureMethod Public Virtual Function Implementations 6c>+≡ (23) <6c 8b>
    bool
    CDLidarCaptureMethod::startTask( CDSimulation * simulation, CDTask * task,
        CDBand * band, long captureCount, const CDDateTime & time )
    {
        // reset the scan index
        this->pulseIndex = 0;

        // check if we should write a separate task header file
        if( this->outputPerPulse ) {

            // create a separate header file
            std::ostringstream hdrFilenameStream;
            hdrFilenameStream << "task_header_" << task->getIndex() << ".dat";
            int hdrFile = open( hdrFilenameStream.str().c_str(),
                O_CREAT | O_WRONLY,
                S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH );

            // check if the open was successful
            if( hdrFile == -1 ) {
                std::cerr
                    << "LidarCaptureMethod::startTask:" << std::endl
                    << "    Could not open output header file!" << std::endl
                    << "    Filename = " << hdrFilenameStream.str() << std::endl;
                exit( 0 );
            }

            // write out the task header data to this file
            this->writeTaskHeader( simulation, band, task, captureCount,
                hdrFile );

            // close the header file
            close( hdrFile );
        }

        // check if we open the output image now
        if( !this->outputPerPulse ) {
            if( !this->openOutputImage( simulation, task, band,
                captureCount, time ) ) {
                exit( 0 );
            }
        }

        return true;
    }

```

Defines:

CDLidarCaptureMethod::startTask(CDSimulation\*,CDTask\*,CDBand\*,long,CDDateTime&),  
never used.

Uses outputPerPulse 5a and pulseIndex 4b.

### 3.7.3 The task end function

This function is called at the end of a task. For this capture method, nothing is done.

8a  $\langle$ CDLidarCaptureMethod Public Virtual Function Declarations 6b $\rangle$ + $\equiv$  (2b)  $\langle$ 6d 8c $\rangle$   
virtual bool endTask( CDSimulation \* simulation, CDTask \* task,  
CDBand \* band, long captureCount, const CDDateTime & time );

8b  $\langle$ CDLidarCaptureMethod Public Virtual Function Implementations 6c $\rangle$ + $\equiv$  (23)  $\langle$ 7 9a $\rangle$   
bool  
CDLidarCaptureMethod::endTask( CDSimulation \* simulation, CDTask \* task,  
CDBand \* band, long captureCount, const CDDateTime & time )  
{  
return true;  
}

Defines:

CDLidarCaptureMethod::startTask(CDSimulation\*,CDTask\*,CDBand\*,long,CDDateTime&),  
never used.

### 3.7.4 The capture start function

This function is called at the start of a capture. For this capture method, we need to make sure to open the image if the image-per-scan option is enabled.

8c  $\langle$ CDLidarCaptureMethod Public Virtual Function Declarations 6b $\rangle$ + $\equiv$  (2b)  $\langle$ 8a 9b $\rangle$   
virtual bool startCapture( CDSimulation \* simulation, CDTask \* task,  
CDBand \* band, long captureCount, const CDDateTime & time );

9a  $\langle$ CDLidarCaptureMethod Public Virtual Function Implementations 6c $\rangle + \equiv$  (23)  $\langle$ 8b 10a $\rangle$

```

bool
CDLidarCaptureMethod::startCapture( CDSimulation * simulation,
    CDTask * task, CDBand * band, long captureCount,
    const CDDateTime & time )
{
    // check if we open the output image now
    if( this->outputPerPulse ) {
        if( !this->openOutputImage( simulation, task, band, 1, time ) ) {
            exit( 0 );
        }
    }

    // shoot the laser into the photon maps
    CDLidarInstrument * lidar =
        static_cast< CDLidarInstrument * >( simulation->getInstrument() );
    simulation->getPhotonMapRepository()->shootPhotons( simulation,
        lidar->getLaser(), time );

    // write the pulse block to the file
    this->writePulseHeader( simulation, band, time,
        band->getOutputImage().getDataFile() );

    return true;
}

```

Defines:  
 CDLidarCaptureMethod::startCapture(CDSimulation\*,CDTask\*,CDBand\*,long,CDDateTime&),  
 never used.  
 Uses outputPerPulse 5a.

### 3.7.5 The capture end function

This function is called at the start of a capture. For this capture method, we need to make sure to close the image if the image-per-scan option is enabled.

9b  $\langle$ CDLidarCaptureMethod Public Virtual Function Declarations 6b $\rangle + \equiv$  (2b)  $\langle$ 8c 10b $\rangle$

```

virtual bool endCapture( CDSimulation * simulation, CDTask * task,
    CDBand * band, long captureCount, const CDDateTime & time );

```

10a  $\langle$ CDLidarCaptureMethod Public Virtual Function Implementations 6c $\rangle$ + $\equiv$  (23)  $\langle$ 9a 10c $\rangle$

```

bool
CDLidarCaptureMethod::endCapture( CDSimulation * simulation, CDTask * task,
    CDBand * band, long captureCount, const CDDateTime & time )
{
    // check if we open the output image now
    if( this->outputPerPulse ) {
        band->getOutputImage().closeDataFile();
    }
    return true;
}

```

Defines:  
 CDLidarCaptureMethod::endCapture(CDSimulation\*,CDTask\*,CDBand\*,long,CDDateTime&),  
 never used.  
 Uses outputPerPulse 5a.

### 3.7.6 The solution computation function

The LIDAR capture method simply creates the solution from the problem everytime.

10b  $\langle$ CDLidarCaptureMethod Public Virtual Function Declarations 6b $\rangle$ + $\equiv$  (2b)  $\langle$ 9b 10d $\rangle$

```

virtual void computeSolution( CDProblem & problem,
    CDSolution & solution );

```

10c  $\langle$ CDLidarCaptureMethod Public Virtual Function Implementations 6c $\rangle$ + $\equiv$  (23)  $\langle$ 10a 11 $\rangle$

```

void
CDLidarCaptureMethod::computeSolution( CDProblem & problem,
    CDSolution & solution )
{
    // create the solution to the current problem
    problem.createSolution( solution, true );

    return;
}

```

Defines:  
 CDLidarCaptureMethod::computeSolution(CDProblem&,CDSolution&), never used.

### 3.7.7 Open the output image file

This is the implementation of the `openOutputImage()` virtual function defined in the `CDCaptureMethod` base class. For LIDAR systems, we pack the temporal dimension into the third dimension instead.

10d  $\langle$ CDLidarCaptureMethod Public Virtual Function Declarations 6b $\rangle$ + $\equiv$  (2b)  $\langle$ 10b 13 $\rangle$

```

bool openOutputImage( CDSimulation * simulation, CDTask * task,
    CDBand * band, long captureCount, const CDDateTime & time );

```

```

11  <CDLidarCaptureMethod Public Virtual Function Implementations 6c>+≡      (23) <10c 14>
      bool
      CDLidarCaptureMethod::openOutputImage( CDSimulation * simulation,
          CDTask * task, CDBand * band, long captureCount,
          const CDDateTime & time )
      {
          // these are the header offsets for the task data and pulse data
          const int taskHeaderOffset = 140;
          const int pulseHeaderOffset = 72;

          // get the output image
          CDEnviImage & outputImage = band->getOutputImage();

          // make a reference copy of the signal gate
          CDSignalGate & signalGate = band->getInstrument()->getSignalGate();

          // setup the output image
          outputImage.setSamples( band->getDetectorArray()->getXElementCount());
          outputImage.setLines(
              band->getDetectorArray()->getYElementCount() * captureCount );
          outputImage.setBands( signalGate.getBinCount());
          outputImage.setDataTypes( CDEnviImage::DoublePrecisionType );
          outputImage.setDescription( "Generated by DIRSIG (LIDAR capture)" );
          outputImage.setOutputBandNames( false );
          outputImage.setOutputBandCenters( true );
          outputImage.setOutputBandWidths( false );

          // the offset depends on if the pulse header data is also written
          if( this->outputPerPulse ) {
              outputImage.setDataOffset( pulseHeaderOffset );
          }
          else {
              outputImage.setDataOffset( taskHeaderOffset + pulseHeaderOffset );
          }

          // set the channel/band names, centers and widths
          for( int ii = 0; ii < signalGate.getBinCount(); ++ii ) {
              outputImage.setBandCenter( ii, signalGate.getTime( ii ) );
          }

          // craft the filename of the image
          std::string imageFilename;
          if( this->outputPerPulse ) {

              // image-per-scan means we have task and pulse indexes in the name
              std::ostringstream imageFilenameStream;

```

```

        imageFilenameStream << band->getOutputImageFilename();
        imageFilenameStream << "-t";
        imageFilenameStream.width( 4 );
        imageFilenameStream.fill( '0' );
        imageFilenameStream << task->getIndex();
        imageFilenameStream.width( 2 );
        imageFilenameStream << "-s";
        imageFilenameStream.width( 4 );
        imageFilenameStream.fill( '0' );
        imageFilenameStream << this->pulseIndex;
        imageFilenameStream << ".img";
        this->pulseIndex += 1;

        imageFilename = imageFilenameStream.str();
    }
    else {
        // normal means we have only the task index name
        std::ostringstream imageFilenameStream;
        imageFilenameStream << band->getOutputImageFilename();
        imageFilenameStream << "-t";
        imageFilenameStream.width( 4 );
        imageFilenameStream.fill( '0' );
        imageFilenameStream << task->getIndex();
        imageFilenameStream << ".img";

        imageFilename = imageFilenameStream.str();
    }

    // write the output header file
    outputImage.setHeaderFilename( imageFilename + ".hdr" );
    outputImage.writeHeaderFile();

    // open the output data file
    outputImage.setDataFilename( imageFilename );
    if( !outputImage.openDataFile( true ) ) {
        std::cerr
            << "LidarCaptureMethod::openOutputImage:" << std::endl
            << "    Could not open output image file!" << std::endl
            << "    Filename = " << imageFilename << std::endl;
        return false;
    }

    // write out the task data to this file
    if( !this->outputPerPulse ) {
        this->writeTaskHeader( simulation, band, task, captureCount,
            outputImage.getDataFile() );
    }

```

```

    }

    return true;
}

```

Defines:

```

CDLidarCaptureMethod::openOutputImage(CDSimulation*, CDTask*, CDBand*, long, CDDateTime&),
    never used.

```

Uses outputPerPulse 5a and pulseIndex 4b.

### 3.7.8 Write the task data to the file

```

13  <CDLidarCaptureMethod Public Virtual Function Declarations 6b>+≡ (2b) <10d 15>
    bool writeTaskHeader( CDSimulation * simulation,
        CDBand * band, CDTask * task, long captureCount, int fd );

```

```

14  <CDLidarCaptureMethod Public Virtual Function Implementations 6c>+≡ (23) <11 16>
    bool
    CDLidarCaptureMethod::writeTaskHeader( CDSimulation * simulation,
        CDBand * band, CDTask * task, long captureCount, int fd )
    {
        // FIXME: get a reference copy of the instrument and mount
        CDLidarInstrument * lidar =
            static_cast< CDLidarInstrument * >( simulation->getInstrument() );

        // FIXME: extract the functional detector array
        CDFunctionalDetectorArray * detectorArray =
            ( CDFunctionalDetectorArray * )band->getDetectorArray();

        // create an instance of the task header
        TaskDataHeader taskHeader;

        // set the ID string
        memset( taskHeader.idString, 64, 0 );
        sprintf( taskHeader.idString, "DIRSIG Lidar Capture: %s", this->version );

        // set the endian (byte order) field
        CDEnviImage::ByteOrder byteOrder = CDEnviImage::getNativeByteOrder();
        taskHeader.endianFlag = 0;
        if( byteOrder == CDEnviImage::BigEndian ) {
            taskHeader.endianFlag = 1;
        }

        // set the file generation date string
        memset( taskHeader.generationDate, 32, 0 );
        time_t currentTime = time( &currentTime );
        struct tm currentTimeTm = *localtime( &currentTime );
        sprintf( taskHeader.generationDate, "%4d%02d%02d%02d%02d",
            1900+currentTimeTm.tm_year,
            currentTimeTm.tm_mon, currentTimeTm.tm_mday,
            currentTimeTm.tm_hour, currentTimeTm.tm_min, currentTimeTm.tm_sec );

        // set the focal length (in meters)
        taskHeader.focalLength = lidar->getFocalLength() * 1e-3;

        // set the focal plane pixel pitches and counts
        taskHeader.xPixelPitch = detectorArray->getXElementSize() * 1e-6;
        taskHeader.yPixelPitch = detectorArray->getYElementSize() * 1e-6;
        taskHeader.xPixelCount = ( unsigned int )detectorArray->getXElementCount();
        taskHeader.yPixelCount = ( unsigned int )detectorArray->getYElementCount();

        // set the laser properties

```

```

taskHeader.pulseRate = 1.0 / lidar->getLaser()->getPulsePeriod();
taskHeader.pulseDuration = lidar->getLaser()->getPulseDuration();
taskHeader.pulsePower = lidar->getLaser()->getPulsePower();
taskHeader.spectralPeak = lidar->getLaser()->getSpectralPeak() * 1e-6;
taskHeader.spectralWidth = lidar->getLaser()->getSpectralWidth() * 1e-6;

// set the time gate start, stop and delta
taskHeader.timeGateStart = lidar->getSignalGate().getMin();
taskHeader.timeGateStop = lidar->getSignalGate().getMax();
taskHeader.timeGateDelta = lidar->getSignalGate().getDelta();
taskHeader.timeBinCount =
    ( unsigned int )lidar->getSignalGate().getBinCount();

// set the number of pulses in the task
taskHeader.pulseCount = captureCount;

// write out the entire task
write( fd, ( void * )&taskHeader, sizeof( taskHeader ) );

return true;
}

```

Defines:

```

CDLidarCaptureMethod::writeTaskHeader(CDSimulation*,CDBand*,CDTask*,long,fstream&),
    never used.

```

Uses TaskDataHeader 3.

### 3.7.9 Write the pulse data header to a file

This function writes a pulse data block to the supplied file. This includes the time pulse was shot, the current platform location and orientation angles, the instrument mount orientation angles, etc.

```

15  <CDLidarCaptureMethod Public Virtual Function Declarations 6b>+≡      (2b) <13 17>
      bool writePulseHeader( CDSimulation * simulation,
                          CDBand * band, const CDDateTime & pulseTime, int fd );

```

```

16  <CDLidarCaptureMethod Public Virtual Function Implementations 6c>+≡ (23) <14 18>
    bool
    CDLidarCaptureMethod::writePulseHeader( CDSimulation * simulation,
        CDBand * band, const CDDateTime & pulseTime, int fd )
    {
        PulseDataHeader pulseHeader;

        // set the pulse time
        pulseHeader.pulseTime = pulseTime.getTotalSeconds();

        // set the platform location
        CDPlatformData platformPosition =
            simulation->getPlatform()->get( pulseTime );
        pulseHeader.xPlatformPosition = platformPosition.getActualLocation().getX();
        pulseHeader.yPlatformPosition = platformPosition.getActualLocation().getY();
        pulseHeader.zPlatformPosition = platformPosition.getActualLocation().getZ();

        // set the platform orientation
        pulseHeader.xPlatformAngle =
            CDGeoBase::radiansToDegrees( platformPosition.getActualXRotation());
        pulseHeader.yPlatformAngle =
            CDGeoBase::radiansToDegrees( platformPosition.getActualYRotation());
        pulseHeader.zPlatformAngle =
            CDGeoBase::radiansToDegrees( platformPosition.getActualZRotation());

        // set the platform along and cross track pointing angles
        double xMountAngle =
            simulation->getInstrument()->getMount()->getXRotation( pulseTime );
        double yMountAngle =
            simulation->getInstrument()->getMount()->getYRotation( pulseTime );
        pulseHeader.alongTrackAngle = CDGeoBase::radiansToDegrees( xMountAngle );
        pulseHeader.acrossTrackAngle = CDGeoBase::radiansToDegrees( yMountAngle );

        // write out the entire pulse header
        write( fd, ( void * )&pulseHeader, sizeof( pulseHeader ) );

        return true;
    }

```

Defines:

```

    CDLidarCaptureMethod::writePulseHeader(CDSimulation*,CDBand*,CDDateTime*), never
    used.

```

Uses PulseDataHeader 4a.

### 3.7.10 The primary function

This class (and all derived classes) are essentially function objects that describe how the spectral radiances in the `CDSolution` are “captured” by the focal plane. Therefore, the `capture()` function is really the only thing described in the class.

```
17  <CDLidarCaptureMethod Public Virtual Function Declarations 6b>+≡ (2b) <15 19>  
    void captureDetector( CDDetector * detector, CDBand * band,  
                          CDSolution & solution );
```

```

18  <CDLidarCaptureMethod Public Virtual Function Implementations 6c>+≡ (23) <16 20>
    void
    CDLidarCaptureMethod::captureDetector( CDDetector * detector, CDBand * band,
        CDSolution & solution )
    {
        // make a reference copy of the signal gate
        CDSignalGate & signalGate = band->getInstrument()->getSignalGate();

        // get the optics throughput of the system
        CDLidarInstrument * lidar =
            static_cast< CDLidarInstrument * >( band->getInstrument() );
        double throughput = lidar->getReceiverTransmission();

        CDSpectralVector< double > photonCounts(
            solution.getSimulation()->getBandpassList(), 0.0 );

        // compute the detector area in m^2 rather than um^2
        double detectorArea = detector->getArea() * 1e-12;

        // print out the result as a function of range
        for( long timeIndex = 0;
            timeIndex < signalGate.getBinCount(); timeIndex += 1 ) {

            // extract the output value
            photonCounts = throughput * solution.getSubPath( 0 )->
                getPhotonCount( timeIndex, detectorArea );

            // check if we are imposing Poisson statistics
            if( this->enablePoissonStatistics ) {

                // compute the Poisson value
                double poissonCount = rng.getPoisson( photonCounts[ 0 ] );

                // stash the value in the pixel buffer
                this->pixelBuffer[ timeIndex ] = poissonCount;
            }
            else {
                // stash the value in the pixel buffer
                this->pixelBuffer[ timeIndex ] = photonCounts[ 0 ];
            }
        }

        // write the output value
        band->getOutputImage().writeData(( char * )this->pixelBuffer,
            signalGate.getBinCount() * sizeof( double ));
    }

```

```

    return;
}

```

Defines:

CDLidarCaptureMethod::capture(CDDetector\*, CDBand\*, CDSolution&), never used.  
 Uses enablePoissonStatistics 5b, pixelBuffer 4c, and rng 5b.

## 3.8 XML Support Functions

### 3.8.1 Load from an XML DOM document

This method reads the state of the object into the supplied XML DOM object.

```

19 <CDLidarCaptureMethod Public Virtual Function Declarations 6b>+≡ (2b) <17 21a>
    virtual bool load( QDomElement & domElement );

```

```

20  <CDLidarCaptureMethod Public Virtual Function Implementations 6c>+≡ (23) <18 21b>
    bool
    CDLidarCaptureMethod::load( QDomElement & domElement )
    {
        // get the "outputfiles" flag
        QDomElement outputElement = getElement( domElement, "outputfiles" );
        if( outputElement.isNull() ) {
            std::cerr << std::endl
                << "LidarCapture::load: Format Error!" << std::endl
                << "    outputfiles element missing!" << std::endl
                << std::endl;
            return false;
        }
        QString outputFlag = outputElement.attribute( "perpulse" );
        if( outputFlag == "1" ) {
            this->outputPerPulse = true;
        }
        else {
            this->outputPerPulse = false;
        }

        // get the Poisson statistics flag
        QDomElement poissonElement = getElement( domElement, "poissonstatistics" );
        if( poissonElement.isNull() ) {
            std::cerr << std::endl
                << "LidarCapture::load: Format Error!" << std::endl
                << "    poissonstatistics element missing!" << std::endl
                << std::endl;
            return false;
        }
        QString enabledAttribute = poissonElement.attribute( "enabled" );
        if( enabledAttribute == "1" ) {
            this->enablePoissonStatistics = true;
        }
        else {
            this->enablePoissonStatistics = false;
        }

        return true;
    }

```

Defines:

CDLidarCaptureMethod::load(QDomElement&), never used.

Uses enablePoissonStatistics 5b and outputPerPulse 5a.

### 3.8.2 Store into an XML DOM document

This method outputs the state of the object into the supplied XML DOM object.

```

21a <CDLidarCaptureMethod Public Virtual Function Declarations 6b>+≡ (2b) <19
    virtual bool store( QDomDocument & doc, QDomElement & parentElem );

21b <CDLidarCaptureMethod Public Virtual Function Implementations 6c>+≡ (23) <20
    bool
    CDLidarCaptureMethod::store( QDomDocument & doc, QDomElement & parentElem )
    {
        // create top level "capturemethod" element
        QDomElement captureElement = doc.createElement( "capturemethod" );
        captureElement.setAttribute( "name", "lidar" );
        parentElem.appendChild( captureElement );

        // create the "outputfiles" element
        QDomElement outputElement = doc.createElement( "outputfiles" );
        if( this->outputPerPulse ) {
            outputElement.setAttribute( "enabled", "1" );
        }
        else {
            outputElement.setAttribute( "enabled", "0" );
        }
        captureElement.appendChild( outputElement );

        // create the "poissonstatistics" element
        QDomElement poissonElement = doc.createElement( "poissonstatistics" );
        if( this->enablePoissonStatistics ) {
            poissonElement.setAttribute( "enabled", "1" );
        }
        else {
            poissonElement.setAttribute( "enabled", "0" );
        }
        captureElement.appendChild( poissonElement );

        return true;
    }

```

Defines:

CDLidarCaptureMethod::store(QDomDocument&,QDomElement&), never used.

Uses enablePoissonStatistics 5b and outputPerPulse 5a.

## 4 Header file

The following target collects the contents of the `CDLidarCaptureMethod.h` header file.

```
22 <CDLidarCaptureMethod.h 22>≡
    //
    // DO NOT EDIT THIS FILE!
    // This file was generated by noweb(1) from the following file:
    // $Id: CDLidarCaptureMethod.nw,v 1.43 2006/03/07 14:35:33 sdbpci Exp $
    //
    #ifndef _CDLIDARCAPTUREMETHOD_H_
    #define _CDLIDARCAPTUREMETHOD_H_
    #include "CDCaptureMethod.h"
    #include "CDRandomNumber.h"

    class CDPhotonMapRepository;

    <CDLidarCaptureMethod Class 2b>

    #endif // _CDLIDARCAPTUREMETHOD_H_
```

## 5 Source file

The following target collects the contents of the `CDLidarCaptureMethod.cpp` source file.

```

23  <CDLidarCaptureMethod.cpp 23>≡
    //
    // DO NOT EDIT THIS FILE!
    // This file was generated by noweb(1) from the following file:
    // $Id: CDLidarCaptureMethod.nw,v 1.43 2006/03/07 14:35:33 sdbpci Exp $
    //
    #include "CDLidarCaptureMethod.h"
    #include "CDSimulation.h"
    #include "CDProblem.h"
    #include "CDSolution.h"
    #include "CDDetector.h"
    #include "CDBand.h"
    #include "CDLidarInstrument.h"
    #include "CDScanMethod.h"
    #include "CDFunctionalDetectorArray.h"
    #include "CDPlatform.h"
    #include "CDPlatformData.h"
    #include "CDGeoBase.h"
    #include "CDTask.h"
    #include "CDPhotonMapRepository.h"
    #include "CDXmlFile.h"
    #include <sstream>
    #include <fstream>
    #include <fcntl.h>
    #include <unistd.h>
    #include <sys/stat.h>
    <Document version 2a>
    <CDLidarCaptureMethod Constructor Implementations 5d>
    <CDLidarCaptureMethod Destructor Implementation 6a>
    <CDLidarCaptureMethod Public Virtual Function Implementations 6c>

```

## 6 Revision History

The following chunk stores the log history from the revision control system. This allows the user to review the revision history from the document itself.

```

24 <Revision History 24>≡
  Revision 1.43  2006/03/07 14:35:33  sdbpci
  Fixed an issue where the X & Y location and rotation of the platform
    written to the output file where transposed which resulting in bad
    geolocations in the fxyz_convertor tool.

  Revision 1.42  2006/03/06 20:35:36  sdbpci
  Added a new ID string at the top of the "task" header so that we can
    identify the files created by this capture method.

  Revision 1.41  2006/02/24 03:47:32  sdbpci
  Fixed the store() function which wasn't actually creating the elements that
    it should.

  Revision 1.40  2006/02/24 03:39:48  sdbpci
  Updated the XML description in the "Overview" section to match what is
    currently supported in the load/store functions.
  Moved the CDRandomNumber instance to a private variable rather than a
    sleazy static variable in the computeDetector() function.

  Revision 1.39  2006/01/30 06:15:30  sdbpci
  Added optional support for imposing Poisson statistics on the photon counts
    output by this capture method.

  Revision 1.38  2006/01/09 20:04:35  sdbpci
  Updated to reflect the swap from CDPlatformPosition to CDPlatformData.

  Revision 1.37  2005/12/19 17:22:52  sdbpci
  Changes resulting from the replacement of CDLidar with CDLidarInstrument

  Revision 1.36  2005/10/24 14:37:42  pxlpci
  Removed CDInstrumentMount.h headers

  Revision 1.35  2005/10/19 15:23:53  sdbpci
  Updated to actually be of some use! The only way to get at this capture
    method is via an XML sensor description.
  This object creates a basic set of header information and has some basic
    XML support to control it.

  Revision 1.34  2005/10/18 20:41:19  sdbpci
  Updated to replace the ITT capture method for the rest of the world. Not

```

working yet, but getting closer.