Appendix B: owsScenarios Reference Manual

0.1

Generated by Doxygen 1.4.5

Tue Oct 11 21:40:42 2005

Contents

Chapter 1

owsScenarios Hierarchical Index

1.1 owsScenarios Class Hierarchy

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

Chapter 2

owsScenarios Class Index

2.1 owsScenarios Class List

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

Chapter 3

owsScenarios Class Documentation

3.1 blankFrameParams Class Reference

Data container for parameters when creating an empty frame.

`#include <blankFrameParams.h>`

Public Member Functions

- blankFrameParams ()
- virtual ~blankFrameParams ()

Public Attributes

- int betaDir
- int numSweeps
- int numPx
- double alpha
- double beta
- double tVal
- double deltaBetaPixel
- double deltaAlphaPixel
- double timeStart
- double timeEnd
- double * xPos
- double * yPos
- double * zPos
- double * posTime
- double * pitch
- double * roll
- double * heading
- int n
- double Xvel1
- double Xvel2
- double Yvel1

- double Yvel2
- double Zvel1
- double Zvel2
- double pitchV1
- double pitchV2
- double rollV1
- double rollV2
- double headV1
- double headV2

3.1.1 Detailed Description

Data container for parameters when creating an empty frame.

This class is a data container for the large amount of parameters that must be passed when creating an empty frame.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 blankFrameParams::blankFrameParams ()

The default constructor.

3.1.2.2 blankFrameParams::~blankFrameParams () [virtual]

The destructor.

3.1.3 Member Data Documentation

3.1.3.1 double blankFrameParams::alpha

The alpha angle value associated with the fibers.

3.1.3.2 double blankFrameParams::beta

The beta angle value associated with the oscillating mirror.

3.1.3.3 int blankFrameParams::betaDir

The direction of movement of the beta direction. 0 = direction one (up), 1 = direction 2 (down).

3.1.3.4 double blankFrameParams::deltaAlphaPixel

The change of alpha per pixel during a frame acquisition.

3.1.3.5 double blankFrameParams::deltaBetaPixel

The change of beta per pixel during a frame acquisition.

3.1.3.6 double* blankFrameParams::heading

A pointer to the heading values along the trajectory.

3.1.3.7 double blankFrameParams::headV1

The velocity of the heading at the beginning of acquisition.

3.1.3.8 double blankFrameParams::headV2

The velocity of the heading at the end of acquisition.

3.1.3.9 int blankFrameParams::n

The number of records in the trajectory.

3.1.3.10 int blankFrameParams::numPx

The total number of fibers.

3.1.3.11 int blankFrameParams::numSweeps

The number of sweeps to be made in the frame (the number of lines scanned).

3.1.3.12 double* blankFrameParams::pitch

A pointer to the pitch values along the trajectory.

3.1.3.13 double blankFrameParams::pitchV1

The velocity of the pitch at the beginning of acquisition.

3.1.3.14 double blankFrameParams::pitchV2

The velocity of the pitch at the end of acquisition.

3.1.3.15 double* blankFrameParams::posTime

A pointer to the time values along the trajectory.

3.1.3.16 double* blankFrameParams::roll

A pointer to the roll values along the trajectory.

3.1.3.17 double blankFrameParams::rollV1

The velocity of the roll at the beginning of acquisition.

3.1.3.18 double blankFrameParams::rollV2

The velocity of the roll at the end of acquisition.

3.1.3.19 double blankFrameParams::timeEnd

The ending time of which a frame is captured.The total duration of the frame capture is then timeEnd - timeStart.

3.1.3.20 double blankFrameParams::timeStart

The start of the time of which a frame is captured. The total duration of the frame capture is then timeEnd - timeStart.

3.1.3.21 double blankFrameParams::tVal

The timeout value in clock counts.

3.1.3.22 double* blankFrameParams::xPos

The x position in a 3D coordinate system.

3.1.3.23 double blankFrameParams::Xvel1

The velocity in the X direction at the beginning of acquisition.

3.1.3.24 double blankFrameParams::Xvel2

The velocity in the X direction at the end of acquisition.

3.1.3.25 double* blankFrameParams::yPos

The y position in a 3D coordinate system.

3.1.3.26 double blankFrameParams::Yvel1

The velocity in the Y direction at the beginning of acquisition.

3.1.3.27 double blankFrameParams::Yvel2

The velocity in the Y direction at the end of acquisition.

3.1.3.28 double* blankFrameParams::zPos

The z position in a 3D coordinate system.

3.1.3.29 double blankFrameParams::Zvel1

The velocity in the Z direction at the beginning of acquisition.

3.1.3.30 double blankFrameParams::Zvel2

The velocity in the Z direction at the end of acquisition.

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

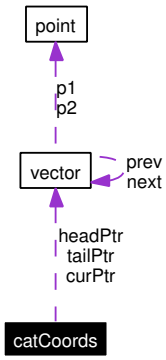
- raytrace1/blankFrameParams.h
- raytrace1/blankFrameParams.cpp

3.2 catCoords Class Reference

Container for coordinates in a catenary object.

`#include <catcoords.h>`

Collaboration diagram for catCoords:



Public Member Functions

- catCoords ()
- ~catCoords ()

  The default destructor.

- void addCoords (vector *vecPtr)

Public Attributes

- vector * headPtr
- vector * tailPtr
- vector * curPtr

3.2.1 Detailed Description

Container for coordinates in a catenary object.

Author:
Rob Russell

3.2.2 Constructor & Destructor Documentation

3.2.2.1 catCoords::catCoords ()

The default constructor.

3.2.3 Member Function Documentation

3.2.3.1 void catCoords::addCoords (vector * *vecPtr*)

Add a particular vector to the set of vectors in this instance.

Parameters:

vecPtr A vector to add to the contained list of vectors.

3.2.4 Member Data Documentation

3.2.4.1 vector* catCoords::curPtr

A scratch pointer for use in the linked list of vectors.

3.2.4.2 vector* catCoords::headPtr

The head of the linked list of vectors.

3.2.4.3 vector* catCoords::tailPtr

The tail of the linked list of vectors.

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

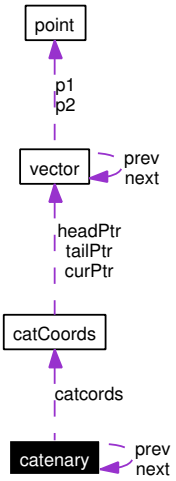
- raytrace1/catcoords.h
- raytrace1/catcoords.cpp

3.3 catenary Class Reference

This class introduces the catenary shape as a basic wire object.

`#include <catenary.h>`

Collaboration diagram for catenary:



Public Member Functions

- catenary ()
- ~catenary ()
- int frameIntersection (frameSet *fr)
- void getParams (wireParams *p)
- double catXatY (double y, double a, double c)
- void segment (point *p)
- void goToHead ()
- void incrementSegment ()
- void RotZ (double phi)
- void RotX (double phi)
- void RotY (double phi)
- void translate (point *p, int convFlag)
- void outputStd ()
- void swapXY ()

Public Attributes

- catCoords * catcords
- float segmentLength
- int numSegments
- float segRemainder
- catenary * prev
- catenary * next

3.3.1 Detailed Description

This class introduces the catenary shape as a basic wire object.

The catenary (a mathematic hyberbolic cosine) describes the shape of a wire hanging under its own weight (or more correctly, a wire or chain affixed at two ends with a uniform acceleration applied to it). This class contains the member functions needed to calculate intersections with itself and pre-generated frames, as well as rotation and translation functions.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 catenary::catenary ()

The default constructor

3.3.2.2 catenary::~catenary ()

The default destructor

3.3.3 Member Function Documentation

3.3.3.1 double catenary::catXatY (double *y*, double *a*, double *c*)

Produces an x value on the catenary given y.

Parameters:

y The y value of interest

a The parameter of the catenary

c The minimum sag minus the parameter of the catenary return The x value of interest

3.3.3.2 int catenary::frameIntersection (frameSet * *fr*)

This is used to calculate the intersections between the catenary and a set of frames. Each segment along the catenary is checked for proximity with each vector in each frame in the set of frames. If they reside close enough (based on the laser footprint, wire radius, etc) and the return energy is high enough, the corresponding vector in the frame is shortened to the location of intersection.

Parameters:

fr The set of frames to be check for intersection.

3.3.3.3 void catenary::getParams (wireParams * *p*)

The wire parameters are set here. The "parameter of the catenary" is solved for using the bisection method.

Parameters:

p The parameters of the wire.

3.3.3.4 void catenary::goToHead ()

Utility used with the linked list operations that makes the current pointer the head pointer.

3.3.3.5 void catenary::incrementSegment ()

Utility used with the linked list operations increases the segment in use to the next one along the catenary.

3.3.3.6 void catenary::outputStd ()

Outputs the catenary points to standard output. This is just a utility function.

3.3.3.7 void catenary::RotX (double *phi*)

Used to rotate the entire catenary (after it has been segmented) about its X axis.

Parameters:
phi The angle of rotation.

3.3.3.8 void catenary::RotY (double *phi*)

Used to rotate the entire catenary (after it has been segmented) about its Y axis.

Parameters:
phi The angle of rotation.

3.3.3.9 void catenary::RotZ (double *phi*)

Used to rotate the entire catenary (after it has been segmented) about its Z axis.

Parameters:
phi The angle of rotation

3.3.3.10 void catenary::segment (point * *p*)

Segment the catenary at a reference point in 3D space

Parameters:
p The point in space to assign to the minimum sag location of the curve. Normally this is (0,0,0).

3.3.3.11 void catenary::swapXY ()

This simply swaps the x and y values of the catenary.

3.3.3.12 void catenary::translate (point * p, int convFlag)

Used to translate the entire catenary (after it has been segmented) to a point in space.

Parameters:

p The point in space to relocate the catenary

convFlag Determines whether or not the catenary coordinates are in UTM or not. If it equals one, then the coordinates are converted to UTM; otherwise, not.

3.3.4 Member Data Documentation

3.3.4.1 catCoords* catenary::catcords

A pointer to the linked list of coordinates that make up the catenary

3.3.4.2 catenary* catenary::next

When used in a linked list, the pointer to the next catenary.

3.3.4.3 int catenary::numSegments

The number of segments created based on segmentLength and the distance between the supports on either end of the catenary.

3.3.4.4 catenary* catenary::prev

When used in a linked list, the pointer to the previous catenary.

3.3.4.5 float catenary::segmentLength

When the catenary is broken into small segments, this variable defines the length of the segments to be created.

3.3.4.6 float catenary::segRemainder

A variable used to keep the remainder of the segment division operation.

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

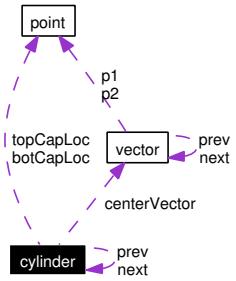
- raytrace1/catenary.h
- raytrace1/catenary.cpp

3.4 cylinder Class Reference

A class that describes a synthetic cylinder in 3 dimensions.

`#include <cylinder.h>`

Collaboration diagram for cylinder:



Public Member Functions

- cylinder ()
- cylinder (point *origin, int originDir, double length, double diameter)
- virtual ~cylinder ()
- void frameIntersection (frameSet *fr)
- void gotoCylHead ()

Public Attributes

- cylinder * prev
- cylinder * next
- point topCapLoc
- point botCapLoc
- double diameter
- double height

3.4.1 Detailed Description

A class that describes a synthetic cylinder in 3 dimensions.

This class is used for poles, trees, and pylons in a synthetic scenario.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 cylinder::cylinder ()

The default constructor

3.4.2.2 cylinder::cylinder (point * *origin*, int *originDir*, double *length*, double *diameter*)

A constructor that creates a cylinder according to the passed parameters.

Parameters:

origin A point that describes the origin of the cylinder

originDir The direction of the cylinder relative to the origin

length The length (in meters) of the cylinder

diameter The diameter of the cylinder

3.4.2.3 cylinder::~cylinder () [virtual]

The default destructor

3.4.3 Member Function Documentation

3.4.3.1 void cylinder::frameIntersection (frameSet * *fr*)

The intersection routine for the cylinder

Parameters:

fr The frame set that is checked for intersections

3.4.3.2 void cylinder::gotoCylHead ()

A linked list utility. It makes the current pointer point to the head of the list.

3.4.4 Member Data Documentation

3.4.4.1 point cylinder::botCapLoc

The point in 3D space of the location of the center of the bottom cap of the cylinder.

3.4.4.2 double cylinder::diameter

The diameter of the cylinder.

3.4.4.3 double cylinder::height

The height of the cylinder.

3.4.4.4 cylinder* cylinder::next

When used in a linked list, this points to the next cylinder in the list.

3.4.4.5 cylinder* cylinder::prev

When used in a linked list, this points to the previous cylinder in the list.

3.4.4.6 point cylinder::topCapLoc

The point in 3D space of the location of the center of the top cap of the cylinder.

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

- owsScenarios/cylinder.h
- owsScenarios/cylinder.cpp

3.5 Ellipsoid Class Reference

Handles latitude/longitude to UTM conversions.

`#include <LatLong-UTMconversion.h>`

Public Member Functions

- Ellipsoid ()
- Ellipsoid (int Id, char *name, double radius, double ecc)

Static Public Member Functions

- static void LLtoUTM (int ReferenceEllipsoid, const double Lat, const double Long, double &UTMNorthing, double &UTMEasting, char *UTMZone)
- static void UTMtoLL (int ReferenceEllipsoid, const double UTMNorthing, const double UTMEasting, const char *UTMZone, double &Lat, double &Long)

Public Attributes

- int id
- char * ellipsoidName
- double EquatorialRadius
- double eccentricitySquared

3.5.1 Detailed Description

Handles latitude/longitude to UTM conversions.

Author:
Rob Russell

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Ellipsoid::Ellipsoid () `[inline]`

The default constructor.

3.5.2.2 Ellipsoid::Ellipsoid (int *Id*, char * *name*, double *radius*, double *ecc*) `[inline]`

Constructor that defines the necessary arguments to do a conversion.

Parameters:

Id Identifies the referenced ellipsoids to be used. The are defined in the file "constants.h", but are also listed here: 1 Airy, 2 Australian National, 3 Bessel 1841, 4 Bessel 1841 (Nambia), 5 Clarke 1866, 6 Clarke 1880, 7 Everest, 8 Fischer 1960 (Mercury), 9 Fischer 1968, 10 GRS 1967, 11 GRS 1980, 12 Helmert 1906, 13 Hough, 14 International, 15 Krassovsky, 16 Modified Airy, 17 Modified Everest, 18 Modified Fischer 1960, 19 South American 1969, 20 WGS 60, 21 WGS 66, 22 WGS-72, 23 WGS-84.

name The name of the Ellipsoid (see the parameter Id).

radius Equatorial Radius of the Ellipsoid.

ecc Eccentricity squared of the Ellipsoid.

3.5.3 Member Function Documentation

3.5.3.1 void Ellipsoid::LLtoUTM (int *ReferenceEllipsoid*, const double *Lat*, const double *Long*, double & *UTMNorthing*, double & *UTMEasting*, char * *UTMZone*) [static]

Used to convert from Latitude/Longitude to UTM.

Parameters:

ReferenceEllipsoid The ID number of the ellipsoid being used.

Lat The latitude to be converted

Long The longitude to be converted

UTMNorthing The northing value returned

UTMEasting The easting value returned

UTMZone The UTM Zone returned

3.5.3.2 void Ellipsoid::UTMtoLL (int *ReferenceEllipsoid*, const double *UTMNorthing*, const double *UTMEasting*, const char * *UTMZone*, double & *Lat*, double & *Long*) [static]

Used to convert from UTM back to Latitude/Longitude

Parameters:

ReferenceEllipsoid The ID number of the ellipsoid being used.

UTMNorthing The northing value to be converted

UTMEasting The easting value to be converted

UTMZone The UTM Zone returned

Lat The latitude returned

Long The longitude returned

3.5.4 Member Data Documentation

3.5.4.1 double Ellipsoid::eccentricitySquared

The eccentricity of the ellipsoid squared.

3.5.4.2 char* Ellipsoid::ellipsoidName

The name of the ellipsoid.

3.5.4.3 double Ellipsoid::EquatorialRadius

The equatorial radius of the ellipsoid.

3.5.4.4 int Ellipsoid::id

The ID number of the ellipsoid being used.

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

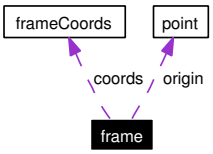
- raytrace1/LatLong-UTMconversion.h
- raytrace1/LatLong-UTMconversion.cpp

3.6 frame Class Reference

Frame Class. This class contains a single frame of data.

`#include <frame.h>`

Collaboration diagram for frame:



Public Member Functions

- frame ()
- virtual ~frame ()
- void generateBlankFrame (blankFrameParams *blfr)
- vector * getBlankFrameGeometry (blankFrameParams *bl)
- void spline (double x[ ], double y[ ], int n, double yp1, double ypn, double y2[ ])
- void splint (double xa[ ], double ya[ ], double y2a[ ], int n, double x, double *y)
- void groundplaneIntersection ()

Public Attributes

- frameCoords * coords
- int numPixels
- int checkFlag
- point origin

3.6.1 Detailed Description

Frame Class. This class contains a single frame of data.

The start origin is specified based on the interpolated position in the frameSet that it belongs to.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 frame::frame ()

The default constructor.

3.6.2.2 frame::~frame () [virtual]

The default destructor.

3.6.3 Member Function Documentation

3.6.3.1 void frame::generateBlankFrame (blankFrameParams * *blfr*)

Generates a blank frame.

Parameters:

blfr is a blankFrameParams pointer.

3.6.3.2 vector * frame::getBlankFrameGeometry (blankFrameParams * *bl*)

Returns a vector from the first to last pixels in a frame.

Parameters:

bl is a blankFrameParams pointer.

3.6.3.3 void frame::groundplaneIntersection ()

Depreciated. Do not use.

3.6.3.4 void frame::spline (double *x*[ ], double *y*[ ], int *n*, double *yp1*, double *ypn*, double *y2*[ ])

Given arrays x[1..n] and y[1..n] containing a tabulated function, i.e., yi = f(xi), with x1 $<$ x2 $<$ . . . $<$ xN , and given values yp1 and ypn for the first derivative of the interpolating function at points 1 and n, respectively, this routine returns an array y2[1..n] that contains the second derivatives of the interpolating function at the tabulated points xi. If yp1 and/or ypn are equal to $1 * 1030$ or larger, the routine is signaled to set the corresponding boundary condition for a natural spline, with zero second derivative on that boundary.

Parameters:

x An array of doubles

y An array of doubles

n The number of data points.

yp1 The first derivative of the interpolating function

ypn The last derivative of the interpolating function

y2 An array of second derivatives of the interpolating function at tabulated points xi

3.6.3.5 void frame::splint (double *xa*[ ], double *ya*[ ], double *y2a*[ ], int *n*, double *x*, double * *y*)

Given the arrays xa[1..n] and ya[1..n], which tabulate a function (with the xai's in order), and given the array y2a[1..n], which is the output from the spline function, and given a value of x, this routine returns a cubic-spline interpolated value y.

Parameters:

xa An array of doubles.

ya An array of doubles.

y2a An array of doubles, output from the spline function.

n The number of data points.

x The given value

y The cubic spline interpolated value

3.6.4 Member Data Documentation

3.6.4.1 int frame::checkFlag

A flag indicating whether the frame should be checked for intersection or not.

3.6.4.2 frameCoords* frame::coords

A pointer to the coordinates of the frame.

3.6.4.3 int frame::numPixels

The number of pixels in this frame.

3.6.4.4 point frame::origin

The origin of the frame.

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

- raytrace1/frame.h
- raytrace1/frame.cpp

3.7 frameCoords Class Reference

A container class for framePoint instances.

`#include <frameCoords.h>`

Public Member Functions

- frameCoords ()
- virtual ~frameCoords ()
- void addCoords (framePoint *vecPtr)

Public Attributes

- std::vector< framePoint > fpoints

  The STL vector object that contains the list of framePoint objects when added.

3.7.1 Detailed Description

A container class for framePoint instances.

Author:
Rob Russell

3.7.2 Constructor & Destructor Documentation

3.7.2.1 frameCoords::frameCoords ()

The default constructor

3.7.2.2 frameCoords::~frameCoords () [virtual]

The default destructor

3.7.3 Member Function Documentation

3.7.3.1 void frameCoords::addCoords (framePoint * *vecPtr*)

For adding framePoint object to the container

Parameters:
vecPtr The framePoint object to be added to the container

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

- raytrace1/frameCoords.h
- raytrace1/frameCoords.cpp

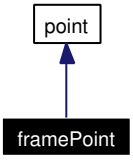
3.8 framePoint Class Reference

A point described in the context of a frame.

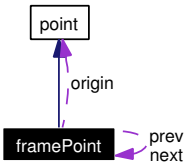
`#include <framePoint.h>`

Inherits point.

Inheritance diagram for framePoint:



Collaboration diagram for framePoint:



Public Member Functions

- framePoint ()
- virtual ~framePoint ()

Public Attributes

- double alpha
- double beta
- unsigned int fiberIndex
- unsigned int scanIndex
- double motorPos
- unsigned int counter
- double time
- point origin
- framePoint * next
- framePoint * prev

3.8.1 Detailed Description

A point described in the context of a frame.

Author:
Rob Russell

3.8.2 Constructor & Destructor Documentation

3.8.2.1 framePoint::framePoint ()

The default constructor

3.8.2.2 framePoint::~framePoint () [virtual]

The default destructor

3.8.3 Member Data Documentation

3.8.3.1 double framePoint::alpha

The alpha value, related to the fiber array, at this point in space

3.8.3.2 double framePoint::beta

The beta value, related to the oscillating mirror, at this point in space.

3.8.3.3 unsigned int framePoint::counter

The counter value for this point. This corresponds to a distance in meters from the origin of the sensor.

3.8.3.4 unsigned int framePoint::fiberIndex

The index of the fiber number for this point

3.8.3.5 double framePoint::motorPos

The position of the motor for this point

3.8.3.6 framePoint* framePoint::next

A pointer to the next framePoint when used in a linked list

3.8.3.7 point framePoint::origin

The point in 3D space of the origin of the sensor

3.8.3.8 framePoint* framePoint::prev

A pointer to the previous framePoint when used in a linked list

3.8.3.9 unsigned int framePoint::scanIndex

The index of the scan number (line scan) for this point

3.8.3.10 double framePoint::time

The time of this point

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

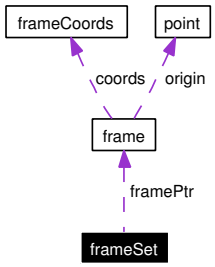
- raytrace1/framePoint.h
- raytrace1/framePoint.cpp

3.9 frameSet Class Reference

A container class that contains a set of frames.

`#include <frameSet.h>`

Collaboration diagram for frameSet:



Public Member Functions

- frameSet ()
- virtual ~frameSet ()
- int addFrame (float timeStart, float timeEnd, navData *navrec, int betaDir)
- vector * getFrameGeometry (float timeStart, float timeEnd, navData *navrec)
- void createStaticNavigation (navData *nav)
- void updateStaticNavigation (navData *nav)
- void insertFrame (frame *fr)
- void goToHead ()
- void incrementFrame ()
- void resetFrame ()
- void incrementCoords ()
- void outputD32Hellas (std::string filename, int type)
- void outputD32Milows (std::string filename, int type)
- void outputD64Milows (std::string filename, int type)
- void outputStandardOut ()
- void groundPlaneIntersect ()
- double horSpacing ()
- double vertSpacing ()

Public Attributes

- double betaFOV
- double alphaFOV
- int numSweeps
- int numPixels
- int numFibers
- double deltaBetaSweep
- double deltaBetaPixel
- double deltaAlphaPixel
- double laserInterval

- double counterResolution
- double mirrorEncAng
- double counterWeight
- unsigned int dropoutVal
- double tVal
- double motorAngInc
- double motorStartAngle
- double laserFreq
- frame * framePtr
- std::vector< frame > vFrames
- int numberOfFrames
- int initializedFlag

3.9.1 Detailed Description

A container class that contains a set of frames.

A set of frames means 1 or more frames that are normally contiguous, but that is not required. Frames are manufactured in this class using a starting time, and ending time, a set of navigation data. The starting and ending times must fall within the time span in the navigation data for the interpolations to work properly.

To create a set of frames, one must call the 'addFrame' function multiple times - once for each frame to be added. That implies that if a user wants to generate frames over a particular time period, e.g. 10 seconds, the time must be incremented by the user for each frame and then a call to addFrame made with the incremented time values.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 frameSet::frameSet ()

The default constructor

3.9.2.2 frameSet::~frameSet () [virtual]

The default destructor

3.9.3 Member Function Documentation

3.9.3.1 int frameSet::addFrame (float *timeStart*, float *timeEnd*, navData * *navrec*, int *betaDir*)

Used for adding a frame to the set of frames contained in this object instance

Parameters:

timeStart The starting time of the frame. It must be within the timespan of the navigation data and less than the timeEnd parameter.

timeEnd The ending time of the frame. It must be within the timespan of the navigation data and greater than the timeStart parameter.

navrec The navigation data to be used for interpolation of the sensor location and orientation along the trajectory.

betaDir The beta direction. 0 = direction 1 (down), 1 = direction 2 (up)

Returns:
TRUE if an error was encountered

3.9.3.2 void frameSet::createStaticNavigation (navData * *nav*)

Creates a local copy of the navigation data. This is necessary for the interpolation routine to work which currently doesn't know how to deal with STL vectors or linked lists; it only works with normal arrays.

Parameters:
nav The navigation data that must be copied

3.9.3.3 vector * frameSet::getFrameGeometry (float *timeStart*, float *timeEnd*, navData * *navrec*)

For retrieving the geometry of the frame, e.g. the 3D coordinates of the first and last points in a given frame. These two points are returned in the form of a vector pointing from the first point to the point furthest away (or the last point).

Parameters:
timeStart The starting time of the frame. It must be within the timespan of the navigation data and less than the timeEnd parameter.

timeEnd The ending time of the frame. It must be within the timespan of the navigation data and greater than the timeStart parameter.

navrec The navigation data to be used for interpolation of the sensor location and orientation along the trajectory.

Returns:
A vector containing the first and last points in a given frame

3.9.3.4 void frameSet::goToHead ()

DEPRECIATED. Was used previously when this class maintained its own linked-list; it has been replaced with an STL vector class.

3.9.3.5 void frameSet::groundPlaneIntersect ()

DEPRECIATED. Was used to calculate ground intersections with frames. Don't use.

3.9.3.6 double frameSet::horSpacing () [inline]

Inline calcuation of the spacing of the beta angle per sweep

3.9.3.7 void frameSet::incrementCoords ()

DEPRECIATED. Was used previously when this class maintained its own linked-list; it has been replaced with an STL vector class.

3.9.3.8 void frameSet::incrementFrame ()

DEPRECIATED. Was used previously when this class maintained its own linked-list; it has been replaced with an STL vector class.

3.9.3.9 void frameSet::insertFrame (frame * *fr*)

Direct insertion of a frame to the frameSet

Parameters:

fr The frame to be added to the set of frames.

3.9.3.10 void frameSet::outputD32Hellas (std::string *filename*, int *type*)

For outputting the data in a frameSet into the 32-bit HELLAS format. The specification for this file type can be found in the EADS/Dornier documentation.

Parameters:

filename The name of the file to be output (full path).

type The type of sensor (currently not used)

3.9.3.11 void frameSet::outputD32Milows (std::string *filename*, int *type*)

For outputting the data in a frameSet into the 32-bit HELLAS format, but using the MilOWS geometric specifications instead. The specification for this sensor and file type can be in EADS/Dornier documentation.

Parameters:

filename The name of the file to be output (full path).

type The type of sensor (currently not used)

3.9.3.12 void frameSet::outputD64Milows (std::string *filename*, int *type*)

For outputting the data in a frameSet into the 64-bit native MilOWS format The specification for this sensor and file type can be in EADS/Dornier documentation.

Parameters:

filename The name of the file to be output (full path).

type The type of sensor (currently not used)

3.9.3.13 void frameSet::outputStandardOut ()

A test function for outputting the frame data to standard output. This is primarily used for testing and making quick plots.

3.9.3.14 void frameSet::resetFrame ()

DEPRECIATED. Was used previously when this class maintained its own linked-list; it has been replaced with an STL vector class.

3.9.3.15 void frameSet::updateStaticNavigation (navData * *nav*)

If the navigation data has changed, then this must be called to update the local copy of it. Please see createStaticNavigation for details.

Parameters:

nav The navigation data that must be updated locally

3.9.3.16 double frameSet::vertSpacing () [inline]

Inline calculation of the spacing of the alpha value per fiber

3.9.4 Member Data Documentation

3.9.4.1 double frameSet::alphaFOV

The total angle associated with the linear fiber array.

3.9.4.2 double frameSet::betaFOV

The total angle swept through by the oscillating mirror in degrees.

3.9.4.3 double frameSet::counterResolution

The resoultion of the counter in seconds.

3.9.4.4 double frameSet::counterWeight

The amount of time per counter tick (in seconds).

3.9.4.5 double frameSet::deltaAlphaPixel

The amount of change of the alpha angle per pixel

3.9.4.6 double frameSet::deltaBetaPixel

The amount of change of the beta angle per pixel.

3.9.4.7 double frameSet::deltaBetaSweep

The amount of change of the beta angle per sweep of the linear array.

3.9.4.8 unsigned int frameSet::dropoutVal

The value, in counts, of the time with wich drop outs occur.

3.9.4.9 frame* frameSet::framePtr

A scratch pointer used to manipulate a frame.

3.9.4.10 int frameSet::initializedFlag

A flag that indicates that the class has already been initialized properly. If equal to one, then the required settings were read from disk correctly; if the value greater than one, then there was an error reading a parameter from an initialization file.

3.9.4.11 double frameSet::laserFreq

The laser frequency.

3.9.4.12 double frameSet::laserInterval

1/the pulse repetition rate of the laser being used.

3.9.4.13 double frameSet::mirrorEncAng

Mirror encoder angle value.

3.9.4.14 double frameSet::motorAngInc

The degrees per pixel that the motor rotates during acquisition.

3.9.4.15 double frameSet::motorStartAngle

The starting angle of the motor.

3.9.4.16 int frameSet::numberOfFrames

The number of frames in this frame set

3.9.4.17 int frameSet::numFibers

The number of fibers in the linear array.

3.9.4.18 int frameSet::numPixels

The total number of pixels per frame.

3.9.4.19 int frameSet::numSweeps

The number of sweeps per frame (the number of line scans per frame).

3.9.4.20 double frameSet::tVal

The value, in time, of the distance that exceeds measuring capability of the sensor.

3.9.4.21 std::vector<frame> frameSet::vFrames

The STL vector container used to contain all of the frames in an instance of frameSet.

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

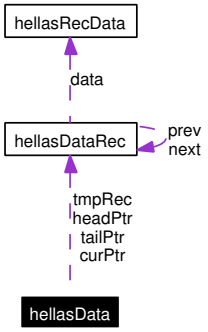
- raytrace1/frameSet.h
- raytrace1/frameSet.cpp

3.10 hellasData Class Reference

Class to handle importing existing HELLAS data.

`#include <hellasData.h>`

Collaboration diagram for hellasData:



Public Member Functions

- hellasData ()
- virtual ~hellasData ()
- int importFile (std::string filename)
- int shiftTimes (navData *nav)

3.10.1 Detailed Description

Class to handle importing existing HELLAS data.

This class can import existing HELLAS data. When successful, it openes an entire file into memory and parses it into readable fields of hellasDataRec records.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 hellasData::hellasData ()

The default constructor.

3.10.2.2 hellasData::~hellasData () [virtual]

The default destructor.

3.10.3 Member Function Documentation

3.10.3.1 int hellasData::importFile (std::string *filename*)

Opens and imports a file into memory.

Parameters:

filename The name of the file to be imported (complete path included).

3.10.3.2 int hellasData::shiftTimes (navData * *nav*)

This function determines whether or not the times in the HELLAS data must be shifted. This occurs when a set of corresponding navigation data has been acquired and the counter being used to time-tag the navigation data and HELLAS records has overflowed. The navigation data is always recorded prior to the HELLAS data, so this can occur.

Parameters:

nav The navigation data to be compared with to determine whether a shift must occur.

Returns:

A return of 0 means the shift did not occur; anything else means it did and was compensated for.

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

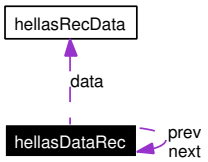
- raytrace1/hellasData.h
- raytrace1/hellasData.cpp

3.11 hellasDataRec Class Reference

A single HELLAS data record.

`#include <hellasDataRec.h>`

Collaboration diagram for hellasDataRec:



Public Member Functions

- hellasDataRec ()
- virtual ~hellasDataRec ()

Public Attributes

- unsigned long rawData
- hellasRecData data
- hellasDataRec * next
- hellasDataRec * prev

3.11.1 Detailed Description

A single HELLAS data record.

Author:
Rob Russell

3.11.2 Constructor & Destructor Documentation

3.11.2.1 hellasDataRec::hellasDataRec ()

The default constructor.

3.11.2.2 hellasDataRec::~hellasDataRec () `[virtual]`

The default destructor.

3.11.3 Member Data Documentation

3.11.3.1 hellasRecData hellasDataRec::data

A container for parsed data from the raw record.

3.11.3.2 hellasDataRec* hellasDataRec::next

A pointer to the next record in a linked list of hellasDataRec records.

3.11.3.3 hellasDataRec* hellasDataRec::prev

A pointer to the previous record in a linked list of hellasDataRec records.

3.11.3.4 unsigned long hellasDataRec::rawData

A data container for a raw data record.

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

- raytrace1/hellasDataRec.h
- raytrace1/hellasDataRec.cpp

3.12 hellasRecData Class Reference

A data container for the actual data contained in a hellasDataRec record.

`#include <hellasRecData.h>`

Public Member Functions

- hellasRecData ()
- virtual ~hellasRecData ()

Public Attributes

- unsigned long counter [HELLAS_A_FRAMESIZE]
- unsigned long timeTag [HELLAS_A_FRAMESIZE]
- unsigned long selectFlag [HELLAS_A_FRAMESIZE]
- unsigned long ztFlag [HELLAS_A_FRAMESIZE]
- unsigned long ainFlag [HELLAS_A_FRAMESIZE]
- unsigned long niFlag [HELLAS_A_FRAMESIZE]
- unsigned long movdirFlag [HELLAS_A_FRAMESIZE]
- unsigned long btFlag [HELLAS_A_FRAMESIZE]
- double realTime [HELLAS_A_FRAMESIZE]

3.12.1 Detailed Description

A data container for the actual data contained in a hellasDataRec record.

The data records that are read in from a HELLAS file are parsed into an instance of this class. The array size HELLAS_A_FRAMESIZE is an external macro and corresponds to the size of a single frame acquisition. Thus, each instance of this class corresponds to a single complete frame.

3.12.2 Constructor & Destructor Documentation

3.12.2.1 hellasRecData::hellasRecData ()

The default constructor.

3.12.2.2 hellasRecData::~hellasRecData () [virtual]

The default destructor.

3.12.3 Member Data Documentation

3.12.3.1 unsigned long hellasRecData::ainFlag[HELLAS_A_FRAMESIZE]

An array containing the ain flag values parsed from a HELLAS data record.

3.12.3.2 unsigned long hellasRecData::btFlag[HELLAS_A_FRAMESIZE]

An array containing the bt flag values parsed from a HELLAS data record.

3.12.3.3 unsigned long hellasRecData::counter[HELLAS_A_FRAMESIZE]

An array containing the counter values parsed from a HELLAS data record.

3.12.3.4 unsigned long hellasRecData::movdirFlag[HELLAS_A_FRAMESIZE]

An array containing the movDir flag values parsed from a HELLAS data record.

3.12.3.5 unsigned long hellasRecData::niFlag[HELLAS_A_FRAMESIZE]

An array containing the ni flag values parsed from a HELLAS data record.

3.12.3.6 double hellasRecData::realTime[HELLAS_A_FRAMESIZE]

The calculated "real time" in second based on the timeTag value. This real time takes into account the counter overflow.

3.12.3.7 unsigned long hellasRecData::selectFlag[HELLAS_A_FRAMESIZE]

An array containing the select flag values parsed from a HELLAS data record.

3.12.3.8 unsigned long hellasRecData::timeTag[HELLAS_A_FRAMESIZE]

An array containing the timetag values parsed from a HELLAS data record.

3.12.3.9 unsigned long hellasRecData::ztFlag[HELLAS_A_FRAMESIZE]

An array containing the zt flag values parsed from a HELLAS data record.

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

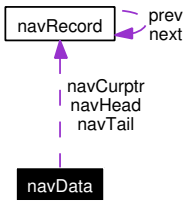
- raytrace1/hellasRecData.h
- raytrace1/hellasRecData.cpp

3.13 navData Class Reference

This class is used to import navigation data.

`#include <navData.h>`

Collaboration diagram for navData:



Public Member Functions

- navData ()
- virtual ~navData ()
- int loadAscFile (std::string filename)
- void outputAll ()
- void setTickWeight (double t)
- void outputUMT ()
- void createStaticNavigation ()

Public Attributes

- int numRecords
- navRecord * navHead
- navRecord * navTail
- navRecord * navCurptr
- int navGenFlag
- double * latPos
- double * lonPos
- double * altPos
- double * pitch
- double * roll
- double * heading
- double * timetag

3.13.1 Detailed Description

This class is used to import navigation data.

The navigation data is in an expected format that matches the format that is exported from HellSim when used with normal HELLAS data. For more information, see the EADS/Dornier documentation.

3.13.2 Constructor & Destructor Documentation

3.13.2.1 navData::navData ()

The default constructor.

3.13.2.2 navData::~navData () [virtual]

The default destructor.

3.13.3 Member Function Documentation

3.13.3.1 void navData::createStaticNavigation ()

A method for creating a static local version of the navigation data. This could be useful, for instance, for plotting with another class that expects an array of data.

3.13.3.2 int navData::loadAscFile (std::string *filename*)

This member function is used to open and parse an ASCII navigation file.

Parameters:
filename The name of the file to be opened, including the path.

Returns:
FALSE the the file was successfully found, opened, and in the expected format as well; else TRUE.

3.13.3.3 void navData::outputAll ()

Used to output the navigation file directly to standard output. Only used for testing.

3.13.3.4 void navData::outputUMT ()

A utility function to output the navigation data in UTM form to a text file.

3.13.3.5 void navData::setTickWeight (double *t*)

The value in time for the clock counts must be defined. HELLAS and MilOWS do not use the same value for each clock tick; therefore, it must be set depending on which sensor recorded the navigation data.

Parameters:
t The time in seconds per clock tick.

3.13.4 Member Data Documentation

3.13.4.1 double* navData::altPos

A pointer to an array of altitude records when a static set of navigation data is created.

3.13.4.2 double* navData::heading

A pointer to an array of heading records when a static set of navigation data is created.

3.13.4.3 double* navData::latPos

A pointer to an array of latitude records when a static set of navigation data is created.

3.13.4.4 double* navData::lonPos

A pointer to an array of longitude records when a static set of navigation data is created.

3.13.4.5 navRecord* navData::navCurptr

A pointer to the iterating pointer when used in a linked list.

3.13.4.6 int navData::navGenFlag

A flag that indicates weather or not the navigation data was statically created or not.

3.13.4.7 navRecord* navData::navHead

A pointer to the head when used in a linked list.

3.13.4.8 navRecord* navData::navTail

A pointer to the tail when used in a linked list.

3.13.4.9 int navData::numRecords

The number of navigation records contained in the instance of this object.

3.13.4.10 double* navData::pitch

A pointer to an array of pitch records when a static set of navigation data is created.

3.13.4.11 double* navData::roll

A pointer to an array of roll records when a static set of navigation data is created.

3.13.4.12 double* navData::timetag

A pointer to an array of time records when a static set of navigation data is created.

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

- raytrace1/navData.h

- raytrace1/navData.cpp

3.14 navRecord Class Reference

A data container for holding a single navigation record.

`#include <navRecord.h>`

Collaboration diagram for navRecord:

Public Member Functions

- navRecord ()
- virtual ~navRecord ()

Public Attributes

- int picNo
- int time
- float head
- float pitch
- float roll
- float vx
- float vy
- float vz
- float vn
- float ve
- float vd
- float v
- float lat
- float lon
- float alt
- double realTime
- navRecord * next
- navRecord * prev

3.14.1 Detailed Description

A data container for holding a single navigation record.

Author:
Rob Russell

3.14.2 Constructor & Destructor Documentation

3.14.2.1 navRecord::navRecord ()

The default constructor

3.14.2.2 navRecord::~navRecord () [virtual]

The default destructor

3.14.3 Member Data Documentation

3.14.3.1 float navRecord::alt

The the current altitude

3.14.3.2 float navRecord::head

The heading value

3.14.3.3 float navRecord::lat

The current latitude

3.14.3.4 float navRecord::lon

The current longitude

3.14.3.5 navRecord* navRecord::next

A pointer to the next navRecord in a linked list.

3.14.3.6 int navRecord::picNo

The picture number

3.14.3.7 float navRecord::pitch

The pitch value

3.14.3.8 navRecord* navRecord::prev

A pointer to the previous navRecord in a linked list.

3.14.3.9 double navRecord::realTime

The computed real time in seconds, taking into account the counter roll-over

3.14.3.10 float navRecord::roll

The roll value

3.14.3.11 int navRecord::time

The time of the record (in clock counts)

3.14.3.12 float navRecord::v

The velocity

3.14.3.13 float navRecord::vd

The downward velocity component

3.14.3.14 float navRecord::ve

The east velocity component

3.14.3.15 float navRecord::vn

The north velocity component

3.14.3.16 float navRecord::vx

The x velocity component

3.14.3.17 float navRecord::vy

The y velocity component

3.14.3.18 float navRecord::vz

The z velocity component

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

- raytrace1/navRecord.h
- raytrace1/navRecord.cpp

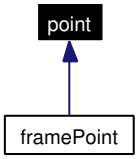
3.15 point Class Reference

A class of a point in 3D.

`#include <point.h>`

Inherited by framePoint.

Inheritance diagram for point:



Public Member Functions

- point ()
- ~point ()
- point * operator+ (point *pt)

Public Attributes

- double x
- double y
- double z

3.15.1 Detailed Description

A class of a point in 3D.

Author:
Rob Russell

3.15.2 Constructor & Destructor Documentation

3.15.2.1 point::point ()

The default constructor.

3.15.2.2 point::~point ()

The default destructor.

3.15.3 Member Function Documentation

3.15.3.1 point * point::operator+ (point * *pt*)

Overloaded operator + . Returns a point.

Parameters:
pt Point to be used during addition

Returns:
Pointer to a point object

3.15.4 Member Data Documentation

3.15.4.1 double point::x

X position in 3D space.

3.15.4.2 double point::y

Y position in 3D space.

3.15.4.3 double point::z

Z position in 3D space.

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

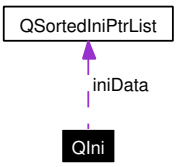
- raytrace1/point.h
- raytrace1/point.cpp

3.16 QIni Class Reference

A class for reading and storing settings from an ini file.

`#include <qini.h>`

Collaboration diagram for QIni:



Public Member Functions

- QIni ()
- QIni (const QString &pathName)
- ~QIni ()
- void setPathName (const QString &pathName)
- QString getPathName ()
- void readIniFile ()
- void writeString (const QString &section, const QString &key, const QString &value)
- QString readString (const QString &section, const QString &key)
- QStringList readSectionValues (const QString &section)
- QStringList readSection (const QString &section)
- void writeInteger (const QString &section, const QString &key, int value)
- int readInteger (const QString &section, const QString &key)
- void writeDouble (const QString &section, const QString &key, double value)
- double readDouble (const QString &section, const QString &key)
- void deleteKey (const QString &section, const QString &key)

Protected Attributes

- QString m_strPathName
- QSortedIniPtrList iniData

3.16.1 Detailed Description

A class for reading and storing settings from an ini file.

Author:
TBScope

Because QSettings isn't really what I was looking for, and because I'm used to using ini files, I created this class. This class creates "standard" ini files.

Here's an example:

[aSection]

aKey=aValue

Note that this class doesn't parse or write comments (at the moment).

3.16.2 Constructor & Destructor Documentation

3.16.2.1 QIni::QIni ()

The constructor. You have to manually set the path.

3.16.2.2 QIni::QIni (const QString & *pathName*)

The recommended constructor.

You can enter the path name via this constructor.

The ini file will automatically be read.

3.16.2.3 QIni::~QIni ()

Destructor.

3.16.3 Member Function Documentation

3.16.3.1 void QIni::deleteKey (const QString & *section*, const QString & *key*)

Delete an entry (key) from a section.

Parameters:

section The section where you want to delete the key from.

key The key you want to delete.

Section and key can be empty. If so, then it will delete only the first occurence of such an entry.

So if you have a list of values, without a sectionname and without keynames, you must call this function every time you want to delete an entry. But this situation is quiet rare I think.

3.16.3.2 QString QIni::getPathName () `[inline]`

Get the current path name.

Returns:

A string containing the current path name.

3.16.3.3 double QIni::readDouble (const QString & *section*, const QString & *key*)

Read a double from the ini file.

Parameters:

section The section where you want to read the string from.

key The key you want to read from.

Returns:
The found value.
If no value could be found, the function will return zero.

Note that both section and key can be empty.

It's not adviced to use this method for empty keynames, use readSectionValues for reading all the values from a certain section.

Key-value pairs that don't belong to a certain section (empty section), are stored on top of the ini file. If you leave the section string empty, it will search those key-value pairs.

3.16.3.4 void QIni::readIniFile ()

Read the ini file and store all records into a pointer list.

3.16.3.5 int QIni::readInteger (const QString & *section*, const QString & *key*)

Read an integer from the ini file.

Parameters:
section The section where you want to read the string from.

key The key you want to read from.

Returns:
The found value.
If no value could be found, the function will return zero.

Note that both section and key can be empty.

It's not adviced to use this method for empty keynames, use readSectionValues for reading all the values from a certain section.

Key-value pairs that don't belong to a certain section (empty section), are stored on top of the ini file. If you leave the section string empty, it will search those key-value pairs.

3.16.3.6 QStringList QIni::readSection (const QString & *section*)

Read all the key=value pairs from a certain section into a stringlist.

Parameters:
section The section you want to read from.

Returns:
A stringlist containing all the values from the given section. If nothing could be found, the function will return an empty stringlist.

Use this function to read stored lists from an ini file.

This function is best used when you also want to read the key names.

Usually, lists are stored without a key name.

3.16.3.7 QStringList QIni::readSectionValues (const QString & *section*)

Read all the values from a certain section into a stringlist.

Parameters:
section The section you want to read from.

Returns:
A stringlist containing all the values from the given section. If nothing could be found, the function will return an empty stringlist.

Use this function to read stored lists from an ini file.

This function is best used when you only want to read the values.

Usually, lists are stored without a key name.

3.16.3.8 QString QIni::readString (const QString & *section*, const QString & *key*)

Read a string from the ini file.

Parameters:
section The section where you want to read the string from.

key The key you want to read from.

Returns:
The found value.
If no value could be found, the function will return an empty string.

Note that both section and key can be empty.

It's not adviced to use this method for empty keynames, use readSectionValues for reading all the values from a certain section.

Key-value pairs that don't belong to a certain section (empty section), are stored on top of the ini file. If you leave the section string empty, it will search those key-value pairs.

3.16.3.9 void QIni::setPathName (const QString & *pathName*) [inline]

Set a new path name.

Parameters:
pathName The new pat hname.

Use this function to open another ini file for example.

The file will automatically be read.

3.16.3.10 void QIni::writeDouble (const QString & *section*, const QString & *key*, double *value*)

Write a double to the ini file.

Parameters:
section The section where you want to store your string in.

key The key name for this string.

value The actual double itself.

Note that both section and key can be empty strings.

If you don't use keys, please use a stringlist and readSectionValues to read all the values. Without a key, it's very hard to know which value you want. And if you don't use a key, it most likely is to store a list of some kind. It's adviced to use a key though.

3.16.3.11 void QIni::writeInteger (const QString & *section*, const QString & *key*, int *value*)

Write an integer to the ini file.

Parameters:

section The section where you want to store your string in.

key The key name for this string.

value The actual integer itself.

Note that both section and key can be empty strings.

If you don't use keys, please use a stringlist and readSectionValues to read all the values. Without a key, it's very hard to know which value you want. And if you don't use a key, it most likely is to store a list of some kind. It's adviced to use a key though.

3.16.3.12 void QIni::writeString (const QString & *section*, const QString & *key*, const QString & *value*)

Write a string to the ini file.

Parameters:

section The section where you want to store your string in.

key The key name for this string.

value The actual string itself.

Note that both section and key can be empty strings.

If you don't use keys, please use a stringlist and readSectionValues to read all the values. Without a key, it's very hard to know which value you want. And if you don't use a key, it most likely is to store a list of some kind. It's adviced to use a key though.

3.16.4 Member Data Documentation

3.16.4.1 QSortedIniPtrList QIni::iniData [protected]

Used internally to store a list of records from the ini file.

3.16.4.2 QString QIni::m_strPathName [protected]

Used internally to store the pathname.

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

- raytrace1/qini.h
- raytrace1/qini.cpp

3.17 QIniData Class Reference

A helper class for storing ini data It's not meant to be used directly, but it's used in QIni to store a pointer list of all ini data.

`#include <qini.h>`

Public Member Functions

- QIniData (const QString &section="", const QString &key="", const QString value="")

3.17.1 Detailed Description

A helper class for storing ini data It's not meant to be used directly, but it's used in QIni to store a pointer list of all ini data.

Author:
TBScope

3.17.2 Constructor & Destructor Documentation

3.17.2.1 QIniData::QIniData (const QString & *section* = "", const QString & *key* = "", const QString *value* = "") [inline]

The constructor for an ini data class.

This class contains a data structure for an ini file.

Parameters:

section The section for a key=value pair.

Each section name is written as [sectionname]

key The key name for a setting

Each key is unique for each section. This means that if you have a key named *myKey* in the section *mySection1*, you can't have a key with the same name in the same section. However, you can have a key with the same name in another section though. So *myKey* can be in *mySection1* but also in section *mySection2*

value The value for a certain setting.

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

- raytrace1/qini.h

3.18 QSortedIniPtrList Class Reference

A helper class for sorting the pointer list with stored ini data It's not meant to be used directly, but it's used in QIni as a pointer list of all ini data.

`#include <qini.h>`

Public Member Functions

- int compareItems (QPtrCollection::Item item1, QPtrCollection::Item item2)

3.18.1 Detailed Description

A helper class for sorting the pointer list with stored ini data It's not meant to be used directly, but it's used in QIni as a pointer list of all ini data.

Author:
TBScope

3.18.2 Member Function Documentation

3.18.2.1 int QSortedIniPtrList::compareItems (QPtrCollection::Item *item1*, QPtrCollection::Item *item2*)

Internally used to compare two items, in order to sort them.

NOTE: Sorting happens with the sectionnames and keynames, not the values.

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

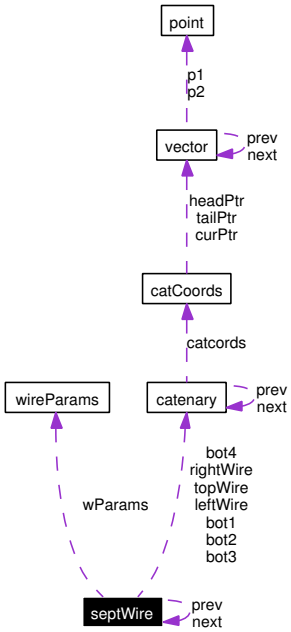
- raytrace1/qini.h
- raytrace1/qini.cpp

3.19 septWire Class Reference

A set of three catenaries.

`#include <septWire.h>`

Collaboration diagram for septWire:



Public Member Functions

- septWire ()
- virtual ~septWire ()
- void createObject (wireParams *params, point *target, float RotX, float RotY, float RotZ)
- void frameIntersection (frameSet *fr)
- point getFirstPoint ()
- point getLastPoint ()

Public Attributes

- septWire * next
- septWire * prev

3.19.1 Detailed Description

A set of three catenaries.

This class contains seven catenaries in a constellation resembling a real world configuration.

3.19.2 Constructor & Destructor Documentation

3.19.2.1 septWire::septWire ()

The default constructor

3.19.2.2 septWire::~septWire () [virtual]

The default desctructor

3.19.3 Member Function Documentation

3.19.3.1 void septWire::createObject (wireParams * *params*, point * *target*, float *RotX*, float *RotY*, float *RotZ*)

Initializes the instance of the septWire object.

Parameters:

params The parameters of the wire

target The target location of the septWire object

RotX The rotation value in degrees about the X axis.

RotY The rotation value in degrees about the Y axis.

RotZ The rotation value in degrees about the Z axis.

3.19.3.2 void septWire::frameIntersection (frameSet * *fr*)

Calculates the intersection with a given frameSet

Parameters:

fr The frameSet to be checked

3.19.3.3 point septWire::getFirstPoint ()

This returns the first point of the top catenary in the septwire object.

Returns:

The first point of the top catenary

3.19.3.4 point septWire::getLastPoint ()

This returns the last point of the top catenary in the septwire object.

Returns:

The last point of the top catenary

3.19.4 Member Data Documentation

3.19.4.1 septWire* septWire::next

For use in a linked list, this points to the next septWire

3.19.4.2 septWire* septWire::prev

For use in a linked list, this points to the previous septWire

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

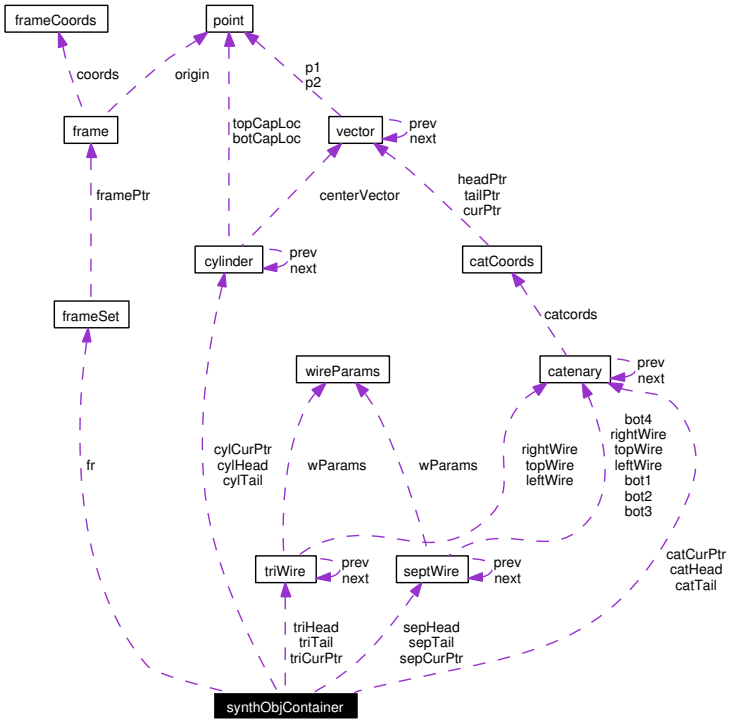
- owsScenarios/septWire.h
- owsScenarios/septWire.cpp

3.20 synthObjContainer Class Reference

This class is a container for all synthetic objects.

`#include <synthObjContainer.h>`

Collaboration diagram for synthObjContainer:



Public Member Functions

- synthObjContainer ()
- virtual ~synthObjContainer ()
- virtual void run ()
- void addObject (catenary *ptr)
- void addObject (cylinder *ptr)
- void addObject (triWire *ptr)
- void addObject (septWire *ptr)
- void delObject (catenary *ptr)
- void delObject (cylinder *ptr)
- void delObject (triWire *ptr)
- void delObject (septWire *ptr)
- void getAllIntersections (frameSet *fr)

Public Attributes

- double percentDone

3.20.1 Detailed Description

This class is a container for all synthetic objects.

It is responsible for containing as well as initiating the intersection routines in all objects within. The advantage of using this class over the contained classes to initiate the intersection routines is that this class spawns a separate thread that calculates while allowing other system events to occur.

3.20.2 Constructor & Destructor Documentation

3.20.2.1 synthObjContainer::synthObjContainer ()

The default constructor

3.20.2.2 synthObjContainer::~synthObjContainer () [virtual]

The default destructor

3.20.3 Member Function Documentation

3.20.3.1 void synthObjContainer::addObject (septWire * *ptr*)

Method to add a septWire object to this container.

Parameters:
ptr The septWire object to add

3.20.3.2 void synthObjContainer::addObject (triWire * *ptr*)

Method to add a triWire object to this container.

Parameters:
ptr The triWire object to add

3.20.3.3 void synthObjContainer::addObject (cylinder * *ptr*)

Method to add a cylinder object to this container.

Parameters:
ptr The cylinder object to add

3.20.3.4 void synthObjContainer::addObject (catenary * *ptr*)

Method to add a catenary object to this container.

Parameters:
ptr The catenary object to add

3.20.3.5 void synthObjContainer::delObject (septWire * ptr)

Method to delete a septWire object from this container.

Parameters:
ptr The septWire object to remove

3.20.3.6 void synthObjContainer::delObject (triWire * ptr)

Method to delete a triWire object from this container.

Parameters:
ptr The triWire object to remove

3.20.3.7 void synthObjContainer::delObject (cylinder * ptr)

Method to delete a cylinder object from this container.

Parameters:
ptr The cylinder object to remove

3.20.3.8 void synthObjContainer::delObject (catenary * ptr)

Method to delete a catenary object from this container.

Parameters:
ptr The catenary object to remove

3.20.3.9 void synthObjContainer::getAllIntersections (frameSet * fr)

A method to calculate the intersections between all of the synthetic objects contained in this class and the specified frameSet.

Parameters:
fr The frameSet to be checked for intersection.

3.20.3.10 void synthObjContainer::run () [virtual]

The overloaded member function that contains the code to be run in a separate thread. However, the user must call the method "start" derived from the QThread class to start, NOT run().

3.20.4 Member Data Documentation

3.20.4.1 double synthObjContainer::percentDone

An estimate on the percentage of processing that is done when calculating intersections.

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

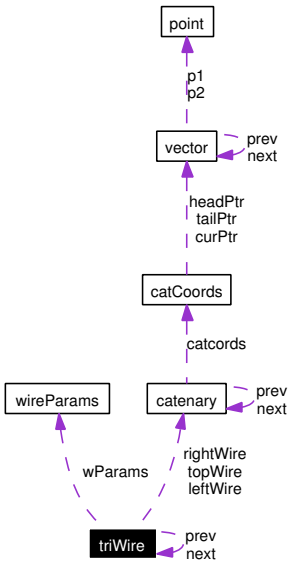
- owsScenarios/synthObjContainer.h
- owsScenarios/synthObjContainer.cpp

3.21 triWire Class Reference

A tri-wire configuration with three catenaries.

`#include <triWire.h>`

Collaboration diagram for triWire:



Public Member Functions

- triWire ()
- virtual ~triWire ()
- void createObject (wireParams *params, point *target, float RotX, float RotY, float RotZ)
- void frameIntersection (frameSet *fr)
- point getFirstPoint ()
- point getLastPoint ()

Public Attributes

- triWire * next
- triWire * prev

3.21.1 Detailed Description

A tri-wire configuration with three catenaries.

This configuration resembles a real-world three wire configuration; it is the same as the top three wires in a seven-wire pylon configuration.

3.21.2 Constructor & Destructor Documentation

3.21.2.1 triWire::triWire ()

The default constructor.

3.21.2.2 triWire::~triWire () [virtual]

The default destructor.

3.21.3 Member Function Documentation

3.21.3.1 void triWire::createObject (wireParams * *params*, point * *target*, float *RotX*, float *RotY*, float *RotZ*)

Initializes the instance of the triWire object.

Parameters:

params The parameters of the wire

target The target location of the triWire object

RotX The rotation value in degrees about the X axis.

RotY The rotation value in degrees about the Y axis.

RotZ The rotation value in degrees about the Z axis.

3.21.3.2 void triWire::frameIntersection (frameSet * *fr*)

Calculates the intersection with a given frameSet

Parameters:

fr The frameSet to be checked

3.21.3.3 point triWire::getFirstPoint ()

This returns the first point of the top catenary in the septwire object.

Returns:

The first point of the top catenary

3.21.3.4 point triWire::getLastPoint ()

This returns the last point of the top catenary in the septwire object.

Returns:

The last point of the top catenary

3.21.4 Member Data Documentation

3.21.4.1 triWire* triWire::next

For use in a linked list, this points to the next septWire

3.21.4.2 triWire* triWire::prev

For use in a linked list, this points to the previous septWire

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

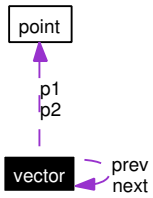
- owsScenarios/triWire.h
- owsScenarios/triWire.cpp

3.22 vector Class Reference

A simple vector class implementation in for vectors in 3D.

`#include <vector.h>`

Collaboration diagram for vector:



Public Member Functions

- vector ()
- ~vector ()
- void clearVector ()
- int LineLineIntersect (vector *v1, vector *v2, point *pa, point *pb, double *mua, double *mub)
- void RotZ (double phi)
- void RotX (double phi)
- void RotY (double phi)

Static Public Member Functions

- static double vecDotVec (vector *v1, vector *v2)
- static double magnitudeVec (vector *v1)
- static double angleVecVec (vector *v1, vector *v2)
- static double cosVecVec (vector *v1, vector *v2)
- static void translateVector (vector *vec, double x, double y, double z)
- static void rotateVector (vector *vec, double alpha, double beta, double gamma)
- static void groundPlaneIntersect (vector *V1)

Public Attributes

- point * p1
- point * p2
- vector * next
- vector * prev
- double eps

3.22.1 Detailed Description

A simple vector class implementation in for vectors in 3D.

Author:
Rob Russell

3.22.2 Constructor & Destructor Documentation

3.22.2.1 vector::vector ()

The default constructor

3.22.2.2 vector::~vector ()

The default destructor

3.22.3 Member Function Documentation

3.22.3.1 double vector::angleVecVec (vector * *v1*, vector * *v2*) [static]

Find the angle between two vectors

Parameters:

v1 The first vector

v2 The second vector

Returns:

The angle between two vectors in space

3.22.3.2 void vector::clearVector ()

Used to clear the values in a vector

3.22.3.3 double vector::cosVecVec (vector * *v1*, vector * *v2*) [static]

Find the cosine of the angle between two vectors

Parameters:

v1 The first vector

v2 The second vector

Returns:

The cosine of the angle between two vectors in space

3.22.3.4 void vector::groundPlaneIntersect (vector * *V1*) [static]

Used to calculate an intersection with the ground plane and shorten the vector accordingly. ** NO LONGER USED **

Parameters:

V1 The vector to be checked and shortened if it intersects with the ground plane

3.22.3.5 int vector::LineLineIntersect (vector * *v1*, vector * *v2*, point * *pa*, point * *pb*, double * *mua*, double * *mub*)

Calculate the shortest segment between two vectors in space.

Parameters:

v1 The first vector

v2 The second vector

pa The point along the vector v1 that is the starting point of the shortest segment

pb The point along the vector v2 that is the ending point of the shortest segment

mua The multiplier with v1 that gives you pa

mub The multiplier with v2 that gives you pb

Returns:

FALSE if computed correctly

3.22.3.6 double vector::magnitudeVec (vector * *v1*) [static]

Computes the magnitude of a single vector

Parameters:

v1 The vector to find the magnitude of

Returns:

The magnitude of the vector

3.22.3.7 void vector::rotateVector (vector * *vec*, double *alpha*, double *beta*, double *gamma*) [static]

For vector rotation

Parameters:

vec The vector to be rotated

alpha Rotation about the first axis

beta Rotation about the second axis

gamma Rotation about the third axis

3.22.3.8 void vector::RotX (double *phi*)

Rotates the vector only about the X axis

Parameters:

phi The angle of rotation about this axis

3.22.3.9 void vector::RotY (double *phi*)

Rotates the vector only about the Y axis

Parameters:
phi The angle of rotation about this axis

3.22.3.10 void vector::RotZ (double *phi*)

Rotates the vector only about the Z axis

Parameters:
phi The angle of rotation about this axis

3.22.3.11 void vector::translateVector (vector * *vec*, double *x*, double *y*, double *z*) [static]

For translating a vector to a new x,y,z coordinate in space

Parameters:
vec The vector to translate
x The new x position in x,y,z coordinates
y The new y position in x,y,z coordinates
z The new z position in x,y,z coordinates

3.22.3.12 double vector::vecDotVec (vector * *v1*, vector * *v2*) [static]

Computes the dot product of two vectors

Parameters:
v1 The first of two vectors
v2 The second of two vectors

Returns:
The vector dot product

3.22.4 Member Data Documentation

3.22.4.1 double vector::eps

The floating point relative accuracy. This is calculated during runtime and is used in the LineLineIntersect function. This should probably be changed to a static or one time calculated value, but it cannot be guaranteed for different machines.

3.22.4.2 vector* vector::next

A pointer to the next vector when used in a linked list.

3.22.4.3 point* vector::p1

The origin of the vector in 3D space

3.22.4.4 point* vector::p2

The end point of the vector in 3D space

3.22.4.5 vector* vector::prev

A pointer to the previous vector when used in a linked list.

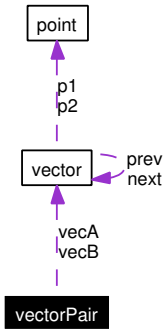
The documentation for this class was generated from the following files:

- raytrace1/vector.h
- raytrace1/vector.cpp

3.23 vectorPair Class Reference

`#include <vectorpair.h>`

Collaboration diagram for vectorPair:



3.23.1 Detailed Description

Author:
Rob Russell

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

- raytrace1/vectorpair.h
- raytrace1/vectorpair.cpp

3.24 wireParams Class Reference

A container for wire creation parameters.

`#include <wireParams.h>`

Public Member Functions

- wireParams ()
- virtual ~wireParams ()

Public Attributes

- float min
- float dist
- float Lheight
- float Rheight
- double albedo
- double seg
- double radius

3.24.1 Detailed Description

A container for wire creation parameters.

These parameters are necessary for the creation of a catenary shaped wire.

3.24.2 Constructor & Destructor Documentation

3.24.2.1 wireParams::wireParams ()

The default constructor

3.24.2.2 wireParams::~wireParams () [virtual]

The default destructor

3.24.3 Member Data Documentation

3.24.3.1 double wireParams::albedo

The reflectivity of the wire, in percentage (0-100%). A value of 0 is not advisable.

3.24.3.2 float wireParams::dist

The distance between the two supports on either end of the wire

3.24.3.3 float wireParams::Lheight

The height of the left support

3.24.3.4 float wireParams::min

The minumum distance of the sag from the theoretical ground.

3.24.3.5 double wireParams::radius

The radius of the wire, in meters.

3.24.3.6 float wireParams::Rheight

The height of the right support

3.24.3.7 double wireParams::seg

The number of segments to break the wire into. This will change the CPU time needed when doing intersection calculation routines.

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

- raytrace1/wireParams.h
- raytrace1/wireParams.cpp

Index