

# Septoplasty Visual Feedback

Generated by Doxygen 1.8.10

Mon May 2 2016 10:07:18



# Contents

<b>1</b>	<b>Namespace Index</b>	<b>1</b>
1.1	Namespace List	1
<b>2</b>	<b>Hierarchical Index</b>	<b>3</b>
2.1	Class Hierarchy	3
<b>3</b>	<b>Class Index</b>	<b>5</b>
3.1	Class List	5
<b>4</b>	<b>Namespace Documentation</b>	<b>7</b>
4.1	animation Namespace Reference	7
4.1.1	Detailed Description	8
4.1.2	Function Documentation	8
4.1.2.1	create_cube_actor(points, epsilon)	8
4.1.2.2	create_training_line(start, end)	8
4.1.2.3	draw_convex_hull(convex_hull)	8
4.1.2.4	get_points_for_cube_from_plane(points, epsilon)	9
4.1.2.5	load_data()	9
4.1.2.6	main()	9
4.1.2.7	mkVtkIdList(it)	9
4.1.2.8	process_line(line)	9
4.1.2.9	process_stdin	9
4.1.3	Variable Documentation	9
4.1.3.1	PACKET_PREFIXES	9
4.2	bufferedReader Namespace Reference	10
4.2.1	Detailed Description	10
4.3	feedback Namespace Reference	10
4.3.1	Detailed Description	10
<b>5</b>	<b>Class Documentation</b>	<b>11</b>
5.1	coordinate Struct Reference	11
5.1.1	Detailed Description	11
5.2	animation.DataCache Class Reference	11

5.2.1	Detailed Description	12
5.3	DataCollectionGroupBox Class Reference	12
5.3.1	Detailed Description	12
5.3.2	Constructor & Destructor Documentation	12
5.3.2.1	DataCollectionGroupBox(QString s)	12
5.3.3	Member Function Documentation	12
5.3.3.1	addControlButton(QWidget *obj)	12
5.3.3.2	addRadioButton(QAbstractButton *obj)	12
5.3.3.3	checkedButton()	13
5.4	DataCollectionWidget Class Reference	13
5.4.1	Detailed Description	13
5.4.2	Member Function Documentation	13
5.4.2.1	clickCalibrationDataRecordGroupBox	13
5.4.2.2	clickOperationalDataRecordGroupBox	13
5.4.2.3	releaseCalibrationCalculationButton	14
5.4.2.4	releaseCalibrationDataRecordGroupBox	14
5.4.2.5	releaseOperationalCalculationButton	14
5.4.2.6	releaseOperationalDataRecordGroupBox	14
5.4.2.7	setModeHandler(ModeHandler *modeHandler)	14
5.5	DataRecordButton Class Reference	14
5.5.1	Detailed Description	14
5.6	DataRecordGroupBox Class Reference	15
5.6.1	Detailed Description	15
5.6.2	Constructor & Destructor Documentation	15
5.6.2.1	DataRecordGroupBox(QString s, bool haveCalculateButton=true)	15
5.6.3	Member Function Documentation	15
5.6.3.1	getCalculationButton()	15
5.6.3.2	getStartStopRecordButton()	15
5.7	FeedbackApplication Class Reference	15
5.7.1	Constructor & Destructor Documentation	16
5.7.1.1	FeedbackApplication(int argc, char **argv)	16
5.8	lineCoordinate Struct Reference	16
5.8.1	Detailed Description	16
5.8.2	Member Function Documentation	16
5.8.2.1	print()	16
5.8.3	Member Data Documentation	17
5.8.3.1	endPoint	17
5.8.3.2	eulerXYZ	17
5.8.3.3	scissorPinchPoint	17
5.8.3.4	startPoint	17

5.8.3.5	timestamp	17
5.8.3.6	valid	17
5.9	lineOfCutGenerator Class Reference	17
5.9.1	Detailed Description	18
5.10	MainWindow Class Reference	18
5.10.1	Detailed Description	18
5.11	ModeHandler Class Reference	18
5.11.1	Constructor & Destructor Documentation	19
5.11.1.1	ModeHandler(DataCollectionWidget *dcw, Platform *platform)	19
5.11.2	Member Function Documentation	19
5.11.2.1	handleCalibrationCalculation()	19
5.11.2.2	handleModeChange(std::string mode, bool toggle)	19
5.11.2.3	handleOperationalCalculation()	19
5.12	bufferedReader.NonBlockingNewlineBufferedReader Class Reference	19
5.12.1	Detailed Description	20
5.13	planeFromLineCoordinate Struct Reference	20
5.13.1	Detailed Description	20
5.13.2	Constructor & Destructor Documentation	20
5.13.2.1	planeFromLineCoordinate(const lineCoordinate &input)	20
5.13.3	Member Function Documentation	21
5.13.3.1	print()	21
5.13.4	Member Data Documentation	21
5.13.4.1	corner_defined	21
5.13.4.2	cornerPoints	21
5.14	Platform Class Reference	21
5.14.1	Detailed Description	22
5.14.2	Member Enumeration Documentation	22
5.14.2.1	DataReceiveMode	22
5.14.3	Member Function Documentation	22
5.14.3.1	call_phantom_add_points()	22
5.14.3.2	call_scissorTrain_add_cut_training_data()	22
5.14.3.3	call_scissorTrain_add_initial_cloud()	22
5.14.3.4	call_scissorTrain_add_line()	22
5.14.3.5	call_scissorTrain_add_scissor_normal_trace_data()	22
5.14.3.6	call_scissorTrain_training_solve(int bladeLength=40, int n_blade_points=1)	23
5.14.3.7	call_septumEstimate_add_nasal_points()	23
5.14.3.8	call_septumEstimate_add_septum_points()	23
5.14.3.9	call_septumEstimate_generate_plane()	23
5.14.3.10	callScissorIncrementActualCutIndex()	23
5.14.3.11	clearCache()	23

5.14.3.12	<code>clearMode()</code>	23
5.14.3.13	<code>isDataValidForReceiveMode(transformComponent2&lt; double &gt; scissorSensor, bool haveScissor, transformComponent2&lt; double &gt; faceSensor, bool haveFace, transformComponent2&lt; double &gt; pointerSensor, bool havePointer)</code>	23
5.14.3.14	<code>loadScissorTrainingData(std::string)</code>	23
5.14.3.15	<code>mainloop(int argc, char **argv)</code>	23
5.14.3.16	<code>openScissorModel()</code>	24
5.14.3.17	<code>receiveMessage(std::istream &amp;stream_)</code>	24
5.14.3.18	<code>saveScissorModel(scissorModel &amp;scissors)</code>	24
5.14.3.19	<code>saveScissorTrainingData(std::string)</code>	24
5.14.3.20	<code>setCacheMode()</code>	24
5.14.3.21	<code>setDataReceiveMode(DataReceiveMode dpm)</code>	24
5.14.3.22	<code>setLineOfCutMode()</code>	24
5.14.3.23	<code>setScissorValidateMode()</code>	24
5.14.3.24	<code>setToolCutValidationMode()</code>	24
5.14.4	Member Data Documentation	25
5.14.4.1	<code>dataReceiveMode</code>	25
5.15	<code>feedback::Properties</code> Class Reference	25
5.15.1	Constructor & Destructor Documentation	25
5.15.1.1	<code>Properties()</code>	25
5.15.2	Member Function Documentation	25
5.15.2.1	<code>getScissorModelFile()</code>	25
5.15.2.2	<code>getSensorSerial(std::string sensor)</code>	25
5.15.2.3	<code>getTrainingDataFile()</code>	25
5.16	<code>scissorModel</code> Struct Reference	26
5.16.1	Detailed Description	26
5.16.2	Member Function Documentation	26
5.16.2.1	<code>print()</code>	26
5.16.3	Member Data Documentation	26
5.16.3.1	<code>bladeLength</code>	26
5.16.3.2	<code>n_blade_points</code>	26
5.16.3.3	<code>normalVector</code>	26
5.16.3.4	<code>scissor_model_id</code>	26
5.16.3.5	<code>sensorTranslationToPinchPoint</code>	26
5.16.3.6	<code>valid</code>	27
5.17	<code>scissorTraining</code> Class Reference	27
5.17.1	Detailed Description	27
5.17.2	Constructor & Destructor Documentation	27
5.17.2.1	<code>scissorTraining()</code>	27
5.17.3	Member Function Documentation	28

5.17.3.1	<code>addCutTrainingData(const std::vector&lt; transformComponent2&lt; double &gt; &gt; &amp;scissor_EM_pose, const std::vector&lt; transformComponent2&lt; double &gt; &gt; &amp;plane_EM_pose)</code> . . . . .	28
5.17.3.2	<code>addInitialCloudData(const std::vector&lt; transformComponent2&lt; double &gt; &gt; &amp;plane_EM_pose_initial_clouds, const std::vector&lt; transformComponent2&lt; double &gt; &gt; &amp;plane_EM_pose)</code> . . . . .	28
5.17.3.3	<code>addLineData(const std::vector&lt; transformComponent2&lt; double &gt; &gt; &amp;line_←point_cloud, const std::vector&lt; transformComponent2&lt; double &gt; &gt; &amp;plane_←EM_pose)</code> . . . . .	28
5.17.3.4	<code>dumpTrainingData()</code> . . . . .	28
5.17.3.5	<code>getAverageInitialPoint(const std::vector&lt; Eigen::Vector3d &gt; &amp;pointer_initial_←clouds)</code> . . . . .	28
5.17.3.6	<code>getLineDirection(const std::vector&lt; Eigen::Vector3d &gt; &amp;line_point_cloud, const std::vector&lt; transformComponent2&lt; double &gt; &gt; &amp;plane_EM_pose, const Eigen::Vector3d &amp;starting_point)</code> . . . . .	28
5.17.3.7	<code>incrementActualCutIndex()</code> . . . . .	28
5.17.3.8	<code>pivotCalibratePinchPoint(const std::vector&lt; transformComponent2&lt; double &gt; &gt; &amp;scissor_EM_pose, const std::vector&lt; transformComponent2&lt; double &gt; &gt; &amp;plane_EM_pose)</code> . . . . .	28
5.17.3.9	<code>printNewTrainingLineData()</code> . . . . .	28
5.17.3.10	<code>printTrainingLineDataAtIndex(std::size_t index_to_process)</code> . . . . .	28
5.17.3.11	<code>setDebugMode(bool mode)</code> . . . . .	29
5.17.3.12	<code>setPlaneNormals(const double plane_normal[4])</code> . . . . .	29
5.17.3.13	<code>setScissorModelForValidation(const scissorModel &amp;input)</code> . . . . .	29
5.17.3.14	<code>setScissorNormal(const double scissorNormal[3])</code> . . . . .	29
5.17.3.15	<code>setTrainingData(const trainingData &amp;input)</code> . . . . .	29
5.17.3.16	<code>solveScissorParameters(const double &amp;bladeLength, const int n_blade_points)</code> . . . . .	29
5.17.3.17	<code>validate(const transformComponent2&lt; double &gt; &amp;pointerEM, const transform←Component2&lt; double &gt; &amp;scissorEM, const transformComponent2&lt; double &gt; &amp;planeEM)</code> . . . . .	29
5.17.3.18	<code>validateWithActualCut(const lineCoordinate &amp;input)</code> . . . . .	29
5.18	<code>septumSurface Struct Reference</code> . . . . .	30
5.18.1	<code>Detailed Description</code> . . . . .	30
5.19	<code>SeptumSurfaceGenerator Class Reference</code> . . . . .	30
5.19.1	<code>Member Function Documentation</code> . . . . .	30
5.19.1.1	<code>addNasalPoints(const std::vector&lt; transformComponent2&lt; double &gt; &gt; &amp;pointer_pose, const std::vector&lt; transformComponent2&lt; double &gt; &gt; &amp;plane_←_EM_pose)</code> . . . . .	30
5.19.1.2	<code>addSeptumPoints(const std::vector&lt; transformComponent2&lt; double &gt; &gt; &amp;pointer_pose, const std::vector&lt; transformComponent2&lt; double &gt; &gt; &amp;plane_←_EM_pose)</code> . . . . .	30
5.19.1.3	<code>generatePhantomPlane()</code> . . . . .	31
5.19.1.4	<code>generatePointCloudFromVector(const std::vector&lt; Eigen::Vector3d &gt; &amp;input, pcl::PointCloud&lt; pcl::PointXYZ &gt;::Ptr &amp;output_cloud)</code> . . . . .	31
5.19.1.5	<code>generateSeptumPlane()</code> . . . . .	31

5.19.1.6	setDistanceThreshold(double distanceThreshold)	31
5.19.2	Member Data Documentation	31
5.19.2.1	haveNasalPoints	31
5.20	bufferedReader.TestNonBlockingNewlineBufferedReader Class Reference	31
5.21	trainingData Struct Reference	32
5.21.1	Detailed Description	32
5.21.2	Member Data Documentation	32
5.21.2.1	cut_directions	32
5.21.2.2	plane_normal	32
5.21.2.3	starting_points	32
5.22	transformComponent< numericStandard > Struct Template Reference	32
5.22.1	Detailed Description	32
5.22.2	Member Data Documentation	33
5.22.2.1	frameNumber	33
5.22.2.2	quaternion	33
5.22.2.3	timestamp	33
5.22.2.4	translation	33
5.23	transformComponent2< numericStandard > Struct Template Reference	33
5.23.1	Detailed Description	33
5.23.2	Member Function Documentation	34
5.23.2.1	setTransform(numericStandard x, numericStandard y, numericStandard z, numericStandard input_rotation[3][3], long timestamp)	34
5.23.3	Member Data Documentation	34
5.23.3.1	frameNumber	34
5.23.3.2	rotation	34
5.23.3.3	timestamp	34
5.23.3.4	translation	34
5.24	animation.VisCache Class Reference	34
5.24.1	Detailed Description	34
5.25	animation.vtkScissorCallback Class Reference	35
5.25.1	Detailed Description	35
5.25.2	Member Function Documentation	35
5.25.2.1	execute(self, obj, event)	35
5.26	animation.vtkTimerCallback Class Reference	35
5.26.1	Detailed Description	36
5.26.2	Constructor & Destructor Documentation	36
5.26.2.1	__init__(self, data_source, renderer)	36
5.26.3	Member Function Documentation	36
5.26.3.1	execute(self, obj, event)	36



[Index](#)

37



# Chapter 1

## Namespace Index

### 1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

<a href="#">animation</a>	7
<a href="#">bufferedReader</a>	10
<a href="#">feedback</a>	10



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

coordinate . . . . .	11
lineCoordinate . . . . .	16
lineOfCutGenerator . . . . .	17
ModeHandler . . . . .	18
object	
animation.DataCache . . . . .	11
animation.VisCache . . . . .	34
bufferedReader.NonBlockingNewlineBufferedReader . . . . .	19
planeFromLineCoordinate . . . . .	20
Platform . . . . .	21
feedback::Properties . . . . .	25
QApplication	
FeedbackApplication . . . . .	15
QGroupBox	
DataCollectionGroupBox . . . . .	12
DataRecordGroupBox . . . . .	15
QMainWindow	
MainWindow . . . . .	18
QPushButton	
DataRecordButton . . . . .	14
QWidget	
DataCollectionWidget . . . . .	13
scissorModel . . . . .	26
scissorTraining . . . . .	27
septumSurface . . . . .	30
SeptumSurfaceGenerator . . . . .	30
TestCase	
bufferedReader.TestNonBlockingNewlineBufferedReader . . . . .	31
trainingData . . . . .	32
transformComponent< numericStandard > . . . . .	32
transformComponent2< numericStandard > . . . . .	33
transformComponent< double > . . . . .	32
animation.vtkScissorCallback . . . . .	35
animation.vtkTimerCallback . . . . .	35



# Chapter 3

## Class Index

### 3.1 Class List

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

<a href="#">coordinate</a>	11
<a href="#">animation.DataCache</a>	11
<a href="#">DataCollectionGroupBox</a>	12
<a href="#">DataCollectionWidget</a>	13
<a href="#">DataRecordButton</a>	14
<a href="#">DataRecordGroupBox</a>	15
<a href="#">FeedbackApplication</a>	15
<a href="#">lineCoordinate</a>	16
<a href="#">lineOfCutGenerator</a>	17
<a href="#">MainWindow</a>	18
<a href="#">ModeHandler</a>	18
<a href="#">bufferedReader.NonBlockingNewlineBufferedReader</a>	19
<a href="#">planeFromLineCoordinate</a>	20
<a href="#">Platform</a>	21
<a href="#">feedback::Properties</a>	25
<a href="#">scissorModel</a>	26
<a href="#">scissorTraining</a>	27
<a href="#">septumSurface</a>	30
<a href="#">SeptumSurfaceGenerator</a>	30
<a href="#">bufferedReader.TestNonBlockingNewlineBufferedReader</a>	31
<a href="#">trainingData</a>	32
<a href="#">transformComponent&lt; numericStandard &gt;</a>	32
<a href="#">transformComponent2&lt; numericStandard &gt;</a>	33
<a href="#">animation.VisCache</a>	34
<a href="#">animation.vtkScissorCallback</a>	35
<a href="#">animation.vtkTimerCallback</a>	35





# Chapter 4

## Namespace Documentation

### 4.1 animation Namespace Reference

#### Classes

- class [DataCache](#)
- class [VisCache](#)
- class [vtkScissorCallback](#)
- class [vtkTimerCallback](#)

#### Functions

- def [process\\_line](#) (line)
- def [process\\_stdin](#)
- def [mkVtkIdList](#) (it)
- def [get\\_points\\_for\\_cube\\_from\\_plane](#) (points, epsilon)
- def [create\\_cube\\_actor](#) (points, epsilon)
- def [load\\_data](#) ()
- def [draw\\_convex\\_hull](#) (convex\_hull)
- def [create\\_training\\_line](#) (start, end)
- def [main](#) ()

#### Variables

- int **FRAME\_RATE** = 15
- int **LINE\_OF\_CUT\_REFRESH\_RATE\_MS** = 20
- int **SCISSOR\_REFRESH\_RATE\_MS** = 20
- int **READ\_DATA\_REFRESH\_RATE\_MS** = 1
- int **UPDATE\_TIMER\_MS** = 50
- list **RED** = [255,0,0]
- list **GREEN** = [0,255,0]
- int **LINE\_OF\_CUT\_LINE\_WIDTH** = 1
- int **TRAINING\_LINE\_WIDTH** = 1
- float **SCISSOR\_OPACITY** = 0.5
- float **PLANE\_OF\_CUT\_EPSILON** = 0.1
- int **DROP\_PACKETS\_OLDER\_THAN\_SEC** = .20
- **TRAINING\_LINE\_COLOR** = GREEN
- **LINE\_OF\_CUT\_COLOR** = RED
- list **SEPTUM\_COLOR** = [240, 200, 201]

- list **PLANE\_OF\_CUT\_COLOR** = [255, 0, 0]
- dictionary **PACKET\_PREFIXES**
- tuple **DIRECTORY** = os.path.dirname(\_\_file\_\_)
- list **threads** = []
- list **exceptions** = []
- **debug** = True
- **EXIT\_APPLICATION** = False

#### 4.1.1 Detailed Description

Visualization of Septum Line of Cut, Surgical Scissors, and Septum.

Uses VTK to visualize line of cut, pipes data from ./feedback

#### 4.1.2 Function Documentation

##### 4.1.2.1 def animation.create\_cube\_actor ( *points*, *epsilon* )

Creates a VTK actor from points and an epsilon (see get\_points\_for\_cube\_from\_plane)

Arguments

```
-----
points : list of four 3-tuples
        precondition : len(points) == 4
epsilon : float
        distance between the two longest faces of the cube that will be
        constructed
```

Returns

```
cube_actor : vtk.vtkActor
        cube that approximates the plane specified by points with width epsilon
```

##### 4.1.2.2 def animation.create\_training\_line ( *start*, *end* )

Creates a training line and returns a vtk.vtkActor representing the training line. The color of the training

Arguments

```
-----
start : 3-tuple of float
        The starting point of the training line
end : 3-tuple of float
        The ending point of the training line
```

Returns

```
-----
training_line_actor : vtk.vtkActor
        an actor specifying a training line from the points start to end
```

##### 4.1.2.3 def animation.draw\_convex\_hull ( *convex\_hull* )

Given a list of points on a convex hull, use vtk.vtkDelaunay2D to draw the convex hull and return a vtk Actor

Arguments

```
-----
convex_hull : list of 3-tuples of float
        points on a convex hull
```

Returns

```
convex_hull_actor : vtk.vtkActor
        VTK Actor modeling a convex_hull
```

#### 4.1.2.4 def animation.get\_points\_for\_cube\_from\_plane ( *points*, *epsilon* )

Give a square representing a plane, construct a cube whose points differ by epsilon.

This allows for the construction of a plane with minimal width for visualization of the plane-of-cut.

Arguments

-----

*points* : list of four 3-tuples  
           precondition : len(*points*) == 4  
*epsilon* : float  
           distance between the two longest faces of the cube that will be constructed

Returns

-----

*cube* : list of eight 3-tuples  
           postcondition : len(*cube*) == 8  
           A cube constructed with two points that have two faces of the same dimension as *points*, and four faces with width of *epsilon*.

#### 4.1.2.5 def animation.load\_data ( )

Deprecated : was formally used to read data from data/Aurora.txt and read the packets in sequentially.

#### 4.1.2.6 def animation.main ( )

Initializes scissor model, line of cut actor, septum plane actor, the camera, the render window, and the line

#### 4.1.2.7 def animation.mkVtkIdList ( *it* )

Creates a VTK ID list from the iterable *it*. This is used for generating planes.

#### 4.1.2.8 def animation.process\_line ( *line* )

Reads a packet (defined in the Packet Routing document) from a string and draws or updates VTK actors with the data

#### 4.1.2.9 def animation.process\_stdin ( *on\_line* = process\_line )

This generates and returns a function that handles reading data from stdin and applies the function *on\_line* to each newline-ending line. The function returned by *process\_stdin* can be used as a VTKTimerCallback function.

Arguments

-----

*on\_line* : callable  
           function will be called on each line of stdin

### 4.1.3 Variable Documentation

#### 4.1.3.1 dictionary animation.PACKET\_PREFIXES

**Initial value:**

```
1 = {
2 #The Packet Prefixes are the first byte in packets received from the Platform
3 #and prediction modules.
4 'SEPTUM_SURFACE_POINTS': '3', #triangles/vertices
5 'SEPTUM_SURFACE_VERTEX_POINTS': '4',
6 'LINE_OF_CUT': '5',
7 'SEPTUM_SURFACE': '6',
8 'TRAINING_LINE': '7',
9 'REMOVE_TRAINING_LINE': '8',
10 'PLANE_OF_CUT': '9',
11 'TRAINING_PLANE': 'A',
12 }
```

## 4.2 bufferedreader Namespace Reference

### Classes

- class [NonBlockingNewlineBufferedReader](#)
- class [TestNonBlockingNewlineBufferedReader](#)

#### 4.2.1 Detailed Description

Author: Michael Norris

Reader for buffering non-blocking reads from stdout that can read multiple lines from stdin and drop all but the newest line. Unit Tests are provided.

## 4.3 feedback Namespace Reference

### Classes

- class [Properties](#)

#### 4.3.1 Detailed Description

[Properties](#) class that reads project properties from properties.xml

## Chapter 5

# Class Documentation

### 5.1 coordinate Struct Reference

```
#include <scissorTraining.h>
```

#### Public Member Functions

- **coordinate** (const Eigen::Vector3d &input)
- Eigen::Vector3d **toEigen3d** () const

#### Public Attributes

- double **xyz** [3]

#### 5.1.1 Detailed Description

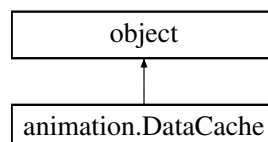
Convenience class for saving training data from Eigen::Vector3d

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

- include/scissorTraining.h

### 5.2 animation.DataCache Class Reference

Inheritance diagram for animation.DataCache:



#### Static Public Attributes

- tuple **lock** = threading.Lock()
- dictionary **cache** = {}

### 5.2.1 Detailed Description

Provides a global view of the data contained in the system, received from packets from the `lineOfCutGenerator` and `septumSurfaceGenerator`

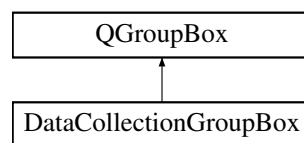
The documentation for this class was generated from the following file:

- `vis/animation.py`

## 5.3 DataCollectionGroupBox Class Reference

```
#include <gui.h>
```

Inheritance diagram for `DataCollectionGroupBox`:



### Public Member Functions

- `DataCollectionGroupBox` (`QString s`)
- `void addRadioButton` (`QAbstractButton *obj`)
- `void addControlButton` (`QWidget *obj`)
- `QAbstractButton * checkedButton` ()

#### 5.3.1 Detailed Description

This provides a `QWidget` that stores radio button for various modes, and a `controlButton` (`DataRecordGroupBox`) that will do things with the selected button (`DataCollectionGroupBox::checkedButton`) when toggled.

The `DataRecordGroupBox`'s actions for the Record button and optional Calculation button are mapped by taking the state of the `DataRecordButton` and the `DataCollectionGroupBox::checkedButton()`, and passing them to the `ModeHandler`, which do things in the `Platform` class with sensor data.

#### 5.3.2 Constructor & Destructor Documentation

##### 5.3.2.1 `DataCollectionGroupBox::DataCollectionGroupBox ( QString s )`

Initialize with `QString s` as the border title.

#### 5.3.3 Member Function Documentation

##### 5.3.3.1 `void DataCollectionGroupBox::addControlButton ( QWidget * obj )`

Adds the control button to the rightmost layout

##### 5.3.3.2 `void DataCollectionGroupBox::addRadioButton ( QAbstractButton * obj )`

Add the radio button to the `radioButtonLayout` and `buttonGroup`

### 5.3.3.3 QAbstractButton \* DataCollectionGroupBox::checkedButton ( )

Returns the button that is currently checked.

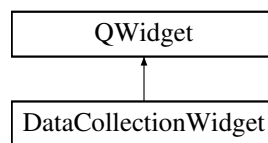
The documentation for this class was generated from the following files:

- include/gui.h
- src/gui.cpp

## 5.4 DataCollectionWidget Class Reference

```
#include <gui.h>
```

Inheritance diagram for DataCollectionWidget:



### Public Slots

- void [releaseOperationalCalculationButton](#) ( )
- void [releaseCalibrationCalculationButton](#) ( )
- void [releaseOperationalDataRecordGroupBox](#) ( )
- void [clickOperationalDataRecordGroupBox](#) (std::string buttonPressed, bool toggle)
- void [releaseCalibrationDataRecordGroupBox](#) ( )
- void [clickCalibrationDataRecordGroupBox](#) (std::string buttonPressed, bool toggle)

### Public Member Functions

- void [setModeHandler](#) (ModeHandler \*modeHandler)

### 5.4.1 Detailed Description

The [DataCollectionWidget](#) contains the Calibration [DataRecordGroupBox](#) and the Operational [DataRecordGroupBox](#). It stacks them on top of each other using a vertical layout. This class also connects a [ModeHandler](#) to the Record and Calculation buttons.

### 5.4.2 Member Function Documentation

5.4.2.1 void [DataCollectionWidget::clickCalibrationDataRecordGroupBox](#) ( std::string *buttonPressed*, bool *toggle* )  
[inline],[slot]

calls [ModeHandler::handleModeChange](#)

5.4.2.2 void [DataCollectionWidget::clickOperationalDataRecordGroupBox](#) ( std::string *buttonPressed*, bool *toggle* )  
[inline],[slot]

calls [ModeHandler::handleModeChange](#)

5.4.2.3 void `DataCollectionWidget::releaseCalibrationCalculationButton` ( ) `[inline],[slot]`

Calls [ModeHandler::handleCalibrationCalculation](#)

5.4.2.4 void `DataCollectionWidget::releaseCalibrationDataRecordGroupBox` ( ) `[inline],[slot]`

calls [DataCollectionWidget::clickCalibrationDataRecordGroupBox](#)

5.4.2.5 void `DataCollectionWidget::releaseOperationalCalculationButton` ( ) `[inline],[slot]`

Calls [ModeHandler::handleOperationalCalculation](#)

5.4.2.6 void `DataCollectionWidget::releaseOperationalDataRecordGroupBox` ( ) `[inline],[slot]`

calls [DataCollectionWidget::clickOperationalDataRecordGroupBox](#)

5.4.2.7 void `DataCollectionWidget::setModeHandler` ( `ModeHandler * modeHandler` )

Sets the [ModeHandler](#) (should be set before the widget is set visible)

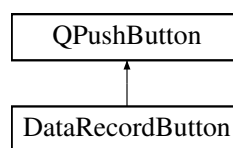
The documentation for this class was generated from the following files:

- include/gui.h
- src/gui.cpp

## 5.5 DataRecordButton Class Reference

```
#include <gui.h>
```

Inheritance diagram for `DataRecordButton`:



### Public Member Functions

- **DataRecordButton** (const `QString` &text, `QWidget` \*parent=0)

#### 5.5.1 Detailed Description

The [DataRecordButton](#) is a button with a title, with the property `QPushButton::setCheckable(true)` set. This allows the button's state to stay toggled.

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

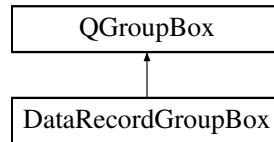
- include/gui.h
- src/gui.cpp



## 5.6 DataRecordGroupBox Class Reference

```
#include <gui.h>
```

Inheritance diagram for DataRecordGroupBox:



### Public Member Functions

- [DataRecordGroupBox](#) (QString s, bool haveCalculateButton=true)
- const [DataRecordButton](#) \* [getStartStopRecordButton](#) ()
- const [QPushButton](#) \* [getCalculationButton](#) ()

#### 5.6.1 Detailed Description

The [DataRecordGroupBox](#) contains a startStop button (That starts and stops a recording based on if the button is pushed or not), and an option calculation button.

#### 5.6.2 Constructor & Destructor Documentation

##### 5.6.2.1 [DataRecordGroupBox::DataRecordGroupBox](#) ( QString s, bool *haveCalculateButton* = true )

Initialize a [DataRecordGroupBox](#) with a Record button and Calculation button.

#### 5.6.3 Member Function Documentation

##### 5.6.3.1 const [QPushButton](#) \* [DataRecordGroupBox::getCalculationButton](#) ( )

Returns the [getCalculationButton](#) if it is present.

##### 5.6.3.2 const [DataRecordButton](#) \* [DataRecordGroupBox::getStartStopRecordButton](#) ( )

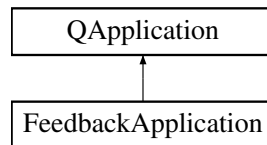
Returns the Record Button

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

- include/gui.h
- src/gui.cpp

## 5.7 FeedbackApplication Class Reference

Inheritance diagram for FeedbackApplication:



## Public Member Functions

- [FeedbackApplication](#) (int argc, char \*\*argv)

### 5.7.1 Constructor & Destructor Documentation

#### 5.7.1.1 [FeedbackApplication::FeedbackApplication \( int argc, char \\*\\* argv \)](#) [inline]

Initialize the Feedback Application, and starts a boost thread that starts [Platform::mainloop](#), which processes data from the SeptoServer application.

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

- include/gui.h

## 5.8 lineCoordinate Struct Reference

```
#include <typedef.h>
```

### Public Member Functions

- void [print](#) ()

### Public Attributes

- double [scissorPinchPoint](#) [3]
- double [eulerXYZ](#) [3]
- double [startPoint](#) [3]
- double [endPoint](#) [3]
- bool [valid](#)
- long [timestamp](#)

#### 5.8.1 Detailed Description

Predicted Line of Cut of the scissors. [lineCoordinate::print](#) will print the Line of Cut packet to be sent to the visualization module.

#### 5.8.2 Member Function Documentation

##### 5.8.2.1 void [lineCoordinate::print \( \)](#) [inline]

Print the line of cut to stdout

### 5.8.3 Member Data Documentation

#### 5.8.3.1 double lineCoordinate::endPoint[3]

the farthest point coordinate where the cut would end on the septum plane coordinate

#### 5.8.3.2 double lineCoordinate::eulerXYZ[3]

Euler Angles

#### 5.8.3.3 double lineCoordinate::scissorPinchPoint[3]

scissor pinch point position relative to the septum plane coordinate

#### 5.8.3.4 double lineCoordinate::startPoint[3]

the point coordinate where the cut would happened on the septum plane coordinate

#### 5.8.3.5 long lineCoordinate::timestamp

unix timestamp in milliseconds of when the SensorData packet was collected.

#### 5.8.3.6 bool lineCoordinate::valid

whether the line coordinate is valid or not. False when the scissor will not cut the septum plane

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

- include/typedef.h

## 5.9 lineOfCutGenerator Class Reference

```
#include <lineOfCutGenerator.h>
```

### Public Member Functions

- **lineOfCutGenerator** (const [septumSurface](#) &surfaceInput, const [scissorModel](#) &scissorInput)
- void **setDebugMode** (bool mode)
- void **setEMdata** (const [transformComponent2](#)< double > &scissor, const [transformComponent2](#)< double > &faceEM)
- void **setEMdata** (const [transformComponent](#)< double > &scissor, const [transformComponent](#)< double > &faceEM)
- void **setSeptumSurface** (const [septumSurface](#) &input)
- void **setScissorModel** (const [scissorModel](#) &input)
- bool **findIntersection** (const Eigen::Vector4d &planeA, const Eigen::Vector4d &planeB, Eigen::Vector3d &pointInPlane, Eigen::Vector3d &intersectionVector)
- Eigen::Vector3d **pointToLineProjection** (const Eigen::Vector3d &pointA, const Eigen::Vector3d &pointInLineB, const Eigen::Vector3d &vectorB)
- Eigen::Vector3d **findLineEndPoint** (const Eigen::Vector3d &pointA, const double &length, const Eigen::Vector3d &vectorB)
- [lineCoordinate](#) **callbackLineOfCut** ()

### 5.9.1 Detailed Description

Takes the pose from the scissor and the face sensors and calculates the line of cut.

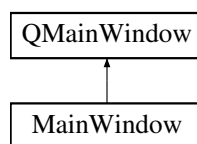
The documentation for this class was generated from the following files:

- include/lineOfCutGenerator.h
- src/lineOfCutGenerator.cpp

### 5.10 MainWindow Class Reference

```
#include <gui.h>
```

Inheritance diagram for MainWindow:



#### Public Member Functions

- **MainWindow** ([Platform](#) \*platform)

#### Public Attributes

- [QVBoxLayout](#) \* **layout**

### 5.10.1 Detailed Description

The Main Window of the application. Contains a [DataCollectionWidget](#) and a menu for loading/saving/clearing training data.

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

- include/gui.h
- src/gui.cpp

### 5.11 ModeHandler Class Reference

#### Public Member Functions

- **ModeHandler** ([DataCollectionWidget](#) \*dcw, [Platform](#) \*platform)
- void [handleModeChange](#) (std::string mode, bool toggle)
- void [handleCalibrationCalculation](#) ()
- void [handleOperationalCalculation](#) ()

### 5.11.1 Constructor & Destructor Documentation

#### 5.11.1.1 ModeHandler::ModeHandler ( DataCollectionWidget \* *dcw*, Platform \* *platform* )

Initialize the [ModeHandler](#).

[ModeHandler](#) handles the mode changes from the gui to the [Platform](#). The gui will create a Mode Handler with a [Platform](#) reference, and the mode handler will handle any callback functions to the [Platform](#).

It should be noted that the [Platform](#) object function calls in this class should be run in a thread, to prevent hanging up the GUI if they take a long time to finish.

### 5.11.2 Member Function Documentation

#### 5.11.2.1 void ModeHandler::handleCalibrationCalculation ( )

Tells the Calibration Module to calculate the scissor model based on the recorded data.

#### 5.11.2.2 void ModeHandler::handleModeChange ( std::string *mode*, bool *toggle* )

Callback from [DataCollectionWidget](#), when the Record button is pressed on or off.

Dispatches calls in the [Platform](#) based on the button that was pressed and whether the Record button was released on or released off.

#### 5.11.2.3 void ModeHandler::handleOperationalCalculation ( )

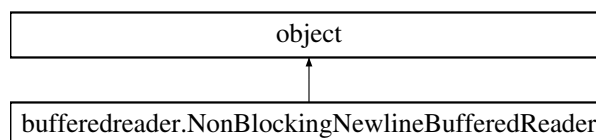
Unused method since there's no calculation that happens in the Operation mode.

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

- include/ModeHandler.h
- src/ModeHandler.cpp

## 5.12 bufferedreader.NonBlockingNewlineBufferedReader Class Reference

Inheritance diagram for bufferedreader.NonBlockingNewlineBufferedReader:



### Public Member Functions

- def **\_\_init\_\_**
- def **read** (self, file\_like)

### Public Attributes

- **last\_read**
- **lines**

- `read_limit`
- `lock`
- `drop_packet_prefixes`

### 5.12.1 Detailed Description

Reader that reads up to `read_limit` characters and will store parts of previous lines.

This will allow you to do non-blocking reads and find lines between reads.

This object reads `stdin`, and either buffers a piece of the most recent line (if less than 1 line is present in the read + what's in the buffer from the last read), return a line if there is only one line in `stdin`, or will return the oldest line that was read from `stdin` (if multiple lines were read at once) that's not also allowed to be dropped.

if more than one line is read, anything after the last newline character will be buffered for the next read.

Usage:

```
reader = NonBlockingBufferedReader(read_limit=100)
out = reader.read(sys.stdin)
```

```
if out:
    #do stuff
    pass
```

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

- `vis/bufferedReader.py`

## 5.13 planeFromLineCoordinate Struct Reference

```
#include <typedef.h>
```

### Public Member Functions

- `planeFromLineCoordinate` (const `lineCoordinate` &input)
- void `print` ()

### Public Attributes

- `lineCoordinate` `line`
- double `cornerPoints` [4][3]
- bool `corner_defined`

#### 5.13.1 Detailed Description

This is a planar representation of the `lineCoordinate`.

#### 5.13.2 Constructor & Destructor Documentation

5.13.2.1 `planeFromLineCoordinate::planeFromLineCoordinate ( const lineCoordinate & input )` `[inline]`

Initialize the plane of cut with a line of cut.

### 5.13.3 Member Function Documentation

#### 5.13.3.1 void planeFromLineCoordinate::print ( ) [inline]

Prints the plane of cut.

### 5.13.4 Member Data Documentation

#### 5.13.4.1 bool planeFromLineCoordinate::corner\_defined

Whether the corners have been specified. If not, then plane's four corners are all the same, [lineCoordinate.scissor←PinchPoint](#)

#### 5.13.4.2 double planeFromLineCoordinate::cornerPoints[4][3]

Corners of the plane

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

- include/typedef.h

## 5.14 Platform Class Reference

```
#include <Platform.h>
```

### Public Types

- enum [DataReceiveMode](#) {  
**PHANTOM\_TRACE, SEPTUM\_TRACE, NASAL\_BRIDGE\_TRACE, CUT,  
 LINE, POINT\_TRACE, TRACE\_SCISSORS, VALIDATE\_SCISSORS** }

### Public Member Functions

- void [receiveMessage](#) (std::istream &stream\_)
- void [mainloop](#) (int argc, char \*\*argv)
- void [clearCache](#) ()
- void [call\\_scissorTrain\\_add\\_line](#) ()
- void [call\\_scissorTrain\\_add\\_initial\\_cloud](#) ()
- void [call\\_scissorTrain\\_add\\_cut\\_training\\_data](#) ()
- void [call\\_scissorTrain\\_add\\_scissor\\_normal\\_trace\\_data](#) ()
- void [call\\_scissorTrain\\_training\\_solve](#) (int bladeLength=40, int n\_blade\_points=1)
- void [call\\_septumEstimate\\_add\\_nasal\\_points](#) ()
- void [call\\_septumEstimate\\_add\\_septum\\_points](#) ()
- void [call\\_septumEstimate\\_generate\\_plane](#) ()
- void [call\\_phantom\\_add\\_points](#) ()
- void [setLineOfCutMode](#) ()
- void [setScissorValidateMode](#) ()
- void [setToolCutValidationMode](#) ()
- void [callScissorIncrementActualCutIndex](#) ()
- void [setCacheMode](#) ()
- void [clearMode](#) ()

- bool `isDataValidForReceiveMode` (`transformComponent2`< double > `scissorSensor`, bool `haveScissor`, `transformComponent2`< double > `faceSensor`, bool `haveFace`, `transformComponent2`< double > `pointer`↔`Sensor`, bool `havePointer`)
- void `setDataReceiveMode` (`DataReceiveMode` `dpm`)
- void `saveScissorModel` (`scissorModel` &`scissors`)
- void `loadScissorTrainingData` (`std::string`)
- void `saveScissorTrainingData` (`std::string`)
- void `openScissorModel` ()

## Public Attributes

- `DataReceiveMode` `dataReceiveMode`
- `feedback::Properties` \* **properties**
- `lineOfCutGenerator` \* **generator**

### 5.14.1 Detailed Description

This class handles incoming messages from an istream, creates the septum surface from calibration, creates the line of cut, and sends outgoing messages to the visualizer.

### 5.14.2 Member Enumeration Documentation

#### 5.14.2.1 enum `Platform::DataReceiveMode`

In `Platform::ProcessingMode.CACHE_MODE` for calibration and Septum Tracing, there are 5 algorithms that can be run. This keeps track of which algorithm will be run when the recording stops.

### 5.14.3 Member Function Documentation

#### 5.14.3.1 void `Platform::call_phantom_add_points` ( )

Add point cloud from tracing the phantom to the `septumSurfaceGenerator` Model. calls `SeptumSurfaceGenerator`↔`::addSeptumPoints`, then `SeptumSurfaceGenerator::generatePhantomPlane`, then `scissorTrainingModel.set`↔`PlaneNormals`

#### 5.14.3.2 void `Platform::call_scissorTrain_add_cut_training_data` ( )

Adds a single scissor cut to the `scissorTrainingModel`

#### 5.14.3.3 void `Platform::call_scissorTrain_add_initial_cloud` ( )

Adds a point cloud to the `scissorTrainingModel`

#### 5.14.3.4 void `Platform::call_scissorTrain_add_line` ( )

Adds a line to the `scissorTrainingModel`

#### 5.14.3.5 void `Platform::call_scissorTrain_add_scissor_normal_trace_data` ( )

Adds the scissor normal from a single trace of the scissors.



5.14.3.6 void Platform::call\_scissorTrain\_training\_solve ( int *bladeLength* = 40, int *n\_blade\_points* = 1 )

Solves for the [scissorModel](#) with the lines, point clouds, cut data, and scissor traces that have been added to the scissorTrainingModel, generates a [scissorModel](#), and saves it to the default scissormodel xml file (scissormodel.xml)

5.14.3.7 void Platform::call\_septumEstimate\_add\_nasal\_points ( )

Add a point cloud from tracing the nasal bridge

5.14.3.8 void Platform::call\_septumEstimate\_add\_septum\_points ( )

Add a point cloud from tracing the septum surface

5.14.3.9 void Platform::call\_septumEstimate\_generate\_plane ( )

Generate the septum plane.

5.14.3.10 void Platform::callScissorIncrementActualCutIndex ( )

Sets the ProcessingMode to CACHE\_MODE (all other modes)

5.14.3.11 void Platform::clearCache ( )

arguments [filename default "data/Aurora.txt"] [-pipe] filename: file to read data from. defaults to data/Auora.txt if not present -pipe: will read data from stdin and ignore file argument. Must be first argument TODO: add slick argparse library

5.14.3.12 void Platform::clearMode ( )

Sets the ProcessingMode to NO\_MODE (drops all packets)

5.14.3.13 bool Platform::isDataValidForReceiveMode ( transformComponent2< double > *scissorSensor*, bool *haveScissor*, transformComponent2< double > *faceSensor*, bool *haveFace*, transformComponent2< double > *pointerSensor*, bool *havePointer* )

The calibration algorithms require vectors of readings from two sensors. Because we allow [Platform](#) to drop packets if the status is not OK, we need to ensure that the data that we have collected for the current frame is appropriate for the DataReceiveMode

5.14.3.14 void Platform::loadScissorTrainingData ( std::string *file* = " " )

Loads the default scissor training data file specified in properties.xml, or loads a scissor model if it's passed as an argument

5.14.3.15 void Platform::mainloop ( int *argc*, char \*\* *argv* )

Reads sensor data from one of 3 modes: pipe, http, readfile.

\$ ./platform pipe This will receive sensor data from stdin and will execute the receiveMessage function with the std::cin istream until it is closed.

\$ ./platform readfile \$FILENAME This will read a recording file from FILENAME and run it through the system. Note that time is not simulated – all packets are thrown through the system in rapid succession.

\$ ./platform http This mode is currently disabled.

#### 5.14.3.16 void Platform::openScissorModel ( )

Attempts to open a ScissorModel from the default scissor model file (scissormodel.xml), and loads it if it exists. If the scissor model file does not exist, then one must be created from calibration before operation mode will run.

#### 5.14.3.17 void Platform::receiveMessage ( std::istream & stream\_ )

Receive a message from the SeptoServer. The format of the message is defined in the document "Packet Routing." In pipe mode, receiveMessage(std::cout) is called, in http mode, a stringstream is used, and to playback a recorded session, an ifstream can be used.

As long as messages are separated by newlines, this can handle receiving multiple messages in a single call, and will continue until the ifstream is closed.

Receives a stream of messages (for reading a file) or a single stream (string stream) from the web service and writes the packet to stdout. Head sensor → port 1 Scissors → port 2 pointer → port 3

#### 5.14.3.18 void Platform::saveScissorModel ( scissorModel & scissors )

Saves the ScissorModel to an xml file

#### 5.14.3.19 void Platform::saveScissorTrainingData ( std::string file = " " )

Saves the scissor training data from the [scissorTraining](#) object to the xml file specified, or if it's not specified, then the default scissor training data file specified in properties.xml will be loaded

#### 5.14.3.20 void Platform::setCacheMode ( )

Changes the ProcessingMode to CACHE\_MODE

#### 5.14.3.21 void Platform::setDataReceiveMode ( DataReceiveMode dpm )

Sets the DataReceiveMode (when ProcessingMode == CACHE\_MODE), which determines what happens to the cached data once the ProcessingMode ends.

#### 5.14.3.22 void Platform::setLineOfCutMode ( )

Sets the ProcessingMode to LINE\_OF\_CUT\_MODE (streaming)

#### 5.14.3.23 void Platform::setScissorValidateMode ( )

Sets the ProcessingMode to SCISSOR\_VALIDATE\_MODE (streaming)

#### 5.14.3.24 void Platform::setToolCutValidationMode ( )

Sets the ProcessingMode to TOOL\_CUT\_VALIDATION\_MODE (streaming)

## 5.14.4 Member Data Documentation

### 5.14.4.1 DataReceiveMode Platform::dataReceiveMode

The current DataReceiveMode. Set by the [ModeHandler](#).

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

- include/Platform.h
- src/Platform.cpp

## 5.15 feedback::Properties Class Reference

### Public Member Functions

- [Properties](#) ()
- std::string [getScissorModelFile](#) ()
- std::string [getSensorSerial](#) (std::string sensor)
- std::string [getTrainingDataFile](#) ()

### Public Attributes

- double **pivot\_offset** [3]

## 5.15.1 Constructor & Destructor Documentation

### 5.15.1.1 feedback::Properties::Properties ( )

[Properties](#) class that reads project properties from properties.xml

## 5.15.2 Member Function Documentation

### 5.15.2.1 std::string feedback::Properties::getScissorModelFile ( )

Returns the name of the file that the scissor model should be saved to /loaded from (defaults to scissormodel.xml)

### 5.15.2.2 std::string feedback::Properties::getSensorSerial ( std::string sensor )

Returns the serial number for the sensor that's in properties.xml, or the string "SENSOR\_IS\_NOT\_DEFINED" if it's not specified.

Since this can return "", `getSensorSerial("phantom").find(parsed_sensor_name)` returns false, but the inverse will always return true since "" is contained in every string.

### 5.15.2.3 std::string feedback::Properties::getTrainingDataFile ( )

Returns the name of the file that the training data should be saved to / loaded from by default (defaults to trainingdata.xml)

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

- include/Properties.h
- src/Properties.cpp

## 5.16 scissorModel Struct Reference

```
#include <typedef.h>
```

### Public Member Functions

- void [print](#) ()

### Public Attributes

- double [bladeLength](#)
- int [scissor\\_model\\_id](#)
- int [n\\_blade\\_points](#)
- double [normalVector](#) [3]
- double [sensorTranslationToPinchPoint](#) [3]
- bool [valid](#)

### 5.16.1 Detailed Description

A model of the scissors, with the translation from the sensor to the scissor pinch point, and the normal of the blades.

### 5.16.2 Member Function Documentation

#### 5.16.2.1 void scissorModel::print ( ) [inline]

Prints the scissor model to stderr

### 5.16.3 Member Data Documentation

#### 5.16.3.1 double scissorModel::bladeLength

the distance from the pinch point to the tip of the blade

#### 5.16.3.2 int scissorModel::n\_blade\_points

The number of blade points. Currently not utilized, but this can allow for scissors with curved blades.

#### 5.16.3.3 double scissorModel::normalVector[3]

Normal of the blades

#### 5.16.3.4 int scissorModel::scissor\_model\_id

Scissor ID (allows for different styles of scissors to have their own scissor model)

#### 5.16.3.5 double scissorModel::sensorTranslationToPinchPoint[3]

Translation from sensor mounted on scissors to the pinch point of the scissors

## 5.16.3.6 bool scissorModel::valid

Whether the model is valid or not. The model needs plane normal, starting point, and line vector.

If valid then the [scissorModel](#) can produce cut data.

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

- include/typedef.h

## 5.17 scissorTraining Class Reference

```
#include <scissorTraining.h>
```

### Public Member Functions

- [scissorTraining](#) ()
- void [setTrainingData](#) (const [trainingData](#) &input)
- void [setPlaneNormals](#) (const double plane\_normal[4])
- void [addLineData](#) (const std::vector< [transformComponent2](#)< double > > &line\_point\_cloud, const std::vector< [transformComponent2](#)< double > > &plane\_EM\_pose)
- void [addInitialCloudData](#) (const std::vector< [transformComponent2](#)< double > > &plane\_EM\_pose\_initial\_clouds, const std::vector< [transformComponent2](#)< double > > &plane\_EM\_pose)
- void [setDebugMode](#) (bool mode)
- Eigen::Vector3d [getLineDirection](#) (const std::vector< Eigen::Vector3d > &line\_point\_cloud, const std::vector< [transformComponent2](#)< double > > &plane\_EM\_pose, const Eigen::Vector3d &starting\_point)
- Eigen::Vector3d [getAverageInitialPoint](#) (const std::vector< Eigen::Vector3d > &pointer\_initial\_clouds)
- void [setScissorNormal](#) (const double scissorNormal[3])
- void [addCutTrainingData](#) (const std::vector< [transformComponent2](#)< double > > &scissor\_EM\_pose, const std::vector< [transformComponent2](#)< double > > &plane\_EM\_pose)
- [scissorModel](#) [solveScissorParameters](#) (const double &bladeLength, const int n\_blade\_points)
- void [setScissorModelForValidation](#) (const [scissorModel](#) &input)
- void [validate](#) (const [transformComponent2](#)< double > &pointerEM, const [transformComponent2](#)< double > &scissorEM, const [transformComponent2](#)< double > &planeEM)
- void [validateWithActualCut](#) (const [lineCoordinate](#) &input)
- void [incrementActualCutIndex](#) ()
- void [printTrainingLineDataAtIndex](#) (std::size\_t index\_to\_process)
- void [printNewTrainingLineData](#) ()
- void [pivotCalibratePinchPoint](#) (const std::vector< [transformComponent2](#)< double > > &scissor\_EM\_pose, const std::vector< [transformComponent2](#)< double > > &plane\_EM\_pose)
- [trainingData](#) [dumpTrainingData](#) ()

### 5.17.1 Detailed Description

Contains data collected from the Training process. Provides functions for calibrating the scissors and verifying the results of the scissor calibration.

When all data is present, [scissorModel](#) [solveScissorParameters](#)(double, int) produces a [scissorModel](#), which predicts the position of the scissor's pinch point from the EM Sensor positioned on the scissor

### 5.17.2 Constructor & Destructor Documentation

#### 5.17.2.1 scissorTraining::scissorTraining ( ) [inline]

Initialize the [scissorTraining](#) object.

### 5.17.3 Member Function Documentation

5.17.3.1 `void scissorTraining::addCutTrainingData ( const std::vector< transformComponent2< double > > & scissor_EM_pose, const std::vector< transformComponent2< double > > & plane_EM_pose )`

Adds a point cloud of the scissor sensor data while it's placed on one of the initial points.

5.17.3.2 `void scissorTraining::addInitialCloudData ( const std::vector< transformComponent2< double > > & plane_EM_pose_initial_clouds, const std::vector< transformComponent2< double > > & plane_EM_pose )`

Adds a line starting point (from placing the EM Pointer on the start of the line and holding it still) to the [scissorTraining](#) model by finding the center of mass of the point cloud.

5.17.3.3 `void scissorTraining::addLineData ( const std::vector< transformComponent2< double > > & line_point_cloud, const std::vector< transformComponent2< double > > & plane_EM_pose )`

Adds line data (from tracing the EM Pointer on a line on the phantom) to the [scissorTraining](#) model.

5.17.3.4 `trainingData scissorTraining::dumpTrainingData ( )`

Turns the [scissorTraining](#) model into a [trainingData](#) object, used for writing the training model in the xml writer for saving trainingdata files.

5.17.3.5 `Eigen::Vector3d scissorTraining::getAverageInitialPoint ( const std::vector< Eigen::Vector3d > & pointer_initial_clouds )`

convert one initial point clouds to a single point by averaging its value

5.17.3.6 `Eigen::Vector3d scissorTraining::getLineDirection ( const std::vector< Eigen::Vector3d > & line_point_cloud, const std::vector< transformComponent2< double > > & plane_EM_pose, const Eigen::Vector3d & starting_point )`

convert one line\_point\_cloud to a single vector using PCA

5.17.3.7 `void scissorTraining::incrementActualCutIndex ( )`

Increments the index of the training line that we validate. This is called when the user selects the Tool Cut Validation mode in the GUI and toggles the Record button.

5.17.3.8 `void scissorTraining::pivotCalibratePinchPoint ( const std::vector< transformComponent2< double > > & scissor_EM_pose, const std::vector< transformComponent2< double > > & plane_EM_pose )`

Performs a pivot calibration on the scissor pinch point.

5.17.3.9 `void scissorTraining::printNewTrainingLineData ( )`

calls `printTrainingLineDataAtIndex` on the newest `cut_direction` vector

5.17.3.10 `void scissorTraining::printTrainingLineDataAtIndex ( std::size_t index_to_process )`

Outputs the training line data and training plane of cut for the index.

5.17.3.11 void scissorTraining::setDebugMode ( bool *mode* )

If true, it will print out a detailed summary of the scissor parameters in realtime while the training mode is activated

5.17.3.12 void scissorTraining::setPlaneNormals ( const double *plane\_normal*[4] )

Sets the plane normal for the phantom surface.

5.17.3.13 void scissorTraining::setScissorModelForValidation ( const scissorModel & *input* )

In validation mode, set the [scissorModel](#). Sets the pinch\_point\_result and scissor\_normal\_result of this [scissorModel](#) from input

5.17.3.14 void scissorTraining::setScissorNormal ( const double *scissorNormal*[3] )

Sets the normal vector of the scissor blade.

5.17.3.15 void scissorTraining::setTrainingData ( const trainingData & *input* )

Set the [trainingData](#) (from loading a training data file), which sets the member variables cut\_directions, starting\_points

5.17.3.16 scissorModel scissorTraining::solveScissorParameters ( const double & *bladeLength*, const int *n\_blade\_points* = 1 )

Produces a [scissorModel](#) from the data stored in the [scissorTraining](#) model.

5.17.3.17 void scissorTraining::validate ( const transformComponent2< double > & *pointerEM*, const transformComponent2< double > & *scissorEM*, const transformComponent2< double > & *planeEM* )

Prints the following data to cerr for each frame received from the sensor: if debugMode Pointer position in scissor coordinates Plane normal in scissor coordinates

always prints: Distance between pinch point and pointer (in mm) Angle between plane normal and scissor plane (in degrees)

5.17.3.18 void scissorTraining::validateWithActualCut ( const lineCoordinate & *input* )

When Platform.procesingMode == Platform::TOOL\_CUT\_VALIDATION\_MODE, this validates each traced line's calibration cut and initial point with the predicted line of cut.

When the user selects the Tool Cut Validation mode in the GUI, and pressed Record in the Calibration mode, the first traced line will be validated. As the Record button is toggled, [scissorTraining::incrementActualCutIndex](#) is called and the next line will be validated.

Prints the following data to cerr for each frame received from the sensor: if debugMode Current target cut initial point Current target cut vector

always prints: Estimated Line of cut vector Estimated scissor pinch point in plane Translation error Rotation error

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

- include/scissorTraining.h
- src/scissorTraining.cpp

## 5.18 septumSurface Struct Reference

```
#include <typedef.h>
```

### Public Attributes

- `std::vector< std::vector< double > >` **convexHull**
- `double` **planeParameters** [4]
- `bool` **valid**

### Friends

- `std::ostream &` **operator<<** (`std::ostream &os`, [septumSurface](#) `const &septum`)

### 5.18.1 Detailed Description

Contains convex hull and plane approximations of the septum surface.

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

- `include/typedef.h`

## 5.19 SeptumSurfaceGenerator Class Reference

### Public Member Functions

- [septumSurface generateSeptumPlane](#) ()
- [septumSurface generatePhantomPlane](#) ()
- `void` [setDistanceThreshold](#) (`double distanceThreshold`)
- `void` [generatePointCloudFromVector](#) (`const std::vector< Eigen::Vector3d > &input`, `pcl::PointCloud< pcl::PointXYZ >::Ptr &output_cloud`)
- `void` [addNasalPoints](#) (`const std::vector< transformComponent2< double > > &pointer_pose`, `const std::vector< transformComponent2< double > > &plane_EM_pose`)
- `void` [addSeptumPoints](#) (`const std::vector< transformComponent2< double > > &pointer_pose`, `const std::vector< transformComponent2< double > > &plane_EM_pose`)

### Public Attributes

- `bool` [haveNasalPoints](#)
- `bool` **haveSeptumPoints**

### 5.19.1 Member Function Documentation

**5.19.1.1** `void SeptumSurfaceGenerator::addNasalPoints ( const std::vector< transformComponent2< double > > &pointer_pose, const std::vector< transformComponent2< double > > &plane_EM_pose )`

Add points that are traced from the outside of the nose on the nasal bridge

**5.19.1.2** `void SeptumSurfaceGenerator::addSeptumPoints ( const std::vector< transformComponent2< double > > &pointer_pose, const std::vector< transformComponent2< double > > &plane_EM_pose )`

Add points that are traced on the septum



## 5.19.1.3 septumSurface SeptumSurfaceGenerator::generatePhantomPlane ( )

Generates the phantom plane

## 5.19.1.4 void SeptumSurfaceGenerator::generatePointCloudFromVector ( const std::vector&lt; Eigen::Vector3d &gt; &amp; input, pcl::PointCloud&lt; pcl::PointXYZ &gt;::Ptr &amp; output\_cloud )

Turns a vector of Eigen points into a PCL point cloud.

## 5.19.1.5 septumSurface SeptumSurfaceGenerator::generateSeptumPlane ( )

Generates the septum plane if we have nasal points and septum points.

## 5.19.1.6 void SeptumSurfaceGenerator::setDistanceThreshold ( double distanceThreshold )

distanceThreshold is the max distance from the scissor pinchpoint to the septum plane that we will calculate a valid line of cut.

## 5.19.2 Member Data Documentation

## 5.19.2.1 bool SeptumSurfaceGenerator::haveNasalPoints

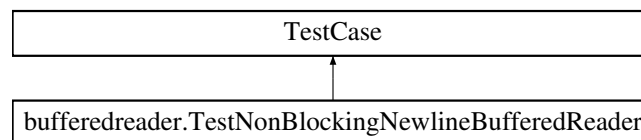
True if we have done the trace nasal bridge and trace septum plane steps in Operational Mode.

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

- include/SeptumSurfaceGenerator.h
- src/SeptumSurfaceGenerator.cpp

## 5.20 bufferedreader.TestNonBlockingNewlineBufferedReader Class Reference

Inheritance diagram for bufferedreader.TestNonBlockingNewlineBufferedReader:



## Public Member Functions

- def **test\_simple\_read** (self)
- def **test\_partial\_read** (self)
- def **test\_multiple\_reads** (self)
- def **test\_read\_limit** (self)

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

- vis/bufferedreader.py

## 5.21 trainingData Struct Reference

```
#include <scissorTraining.h>
```

### Public Attributes

- `std::vector< coordinate > cut_directions`
- `std::vector< coordinate > starting_points`
- `coordinate plane_normal`

### 5.21.1 Detailed Description

Contains all of the data from the Training process.

### 5.21.2 Member Data Documentation

#### 5.21.2.1 `std::vector<coordinate > trainingData::cut_directions`

Contains n `cut_directions` corresponding to `starting_points`. Each `cut_direction` is estimated from tracing the EM Pointer.

#### 5.21.2.2 `coordinate trainingData::plane_normal`

The `plane_normal` corresponds to the normal of the phantom plane, achieved by fitting a plane to a point cloud of data from tracing the EM Pointer on the phantom

#### 5.21.2.3 `std::vector<coordinate > trainingData::starting_points`

Contains n `starting_points` corresponding to `cut_direction`. Each `starting_point` is estimated from averaging the EM Pointer's position while held at a starting point.

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

- `include/scissorTraining.h`

## 5.22 transformComponent< numericStandard > Struct Template Reference

```
#include <typedef.h>
```

### Public Attributes

- `numericStandard quaternion` [4]
- `numericStandard translation` [3]
- `long timestamp`
- `long frameNumber`

### 5.22.1 Detailed Description

```
template<typename numericStandard>struct transformComponent< numericStandard >
```

Contains a Frame Transformation with a quaternion.

## 5.22.2 Member Data Documentation

### 5.22.2.1 `template<typename numericStandard> long transformComponent< numericStandard >::frameNumber`

The frame number of the sensor.

### 5.22.2.2 `template<typename numericStandard> numericStandard transformComponent< numericStandard >::quaternion[4]`

quaternion representation of the rotation matrix of the frame transformation

### 5.22.2.3 `template<typename numericStandard> long transformComponent< numericStandard >::timestamp`

unix timestamp in milliseconds

### 5.22.2.4 `template<typename numericStandard> numericStandard transformComponent< numericStandard >::translation[3]`

translation vector of the frame transformation

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

- include/typedef.h

## 5.23 transformComponent2< numericStandard > Struct Template Reference

```
#include <typedef.h>
```

### Public Member Functions

- void `setTransform` (numericStandard x, numericStandard y, numericStandard z, numericStandard input\_↔ rotation[3][3], long `timestamp`)
- void `setPivotOffset` (numericStandard offset[3])
- void `setPivotOffset` (numericStandard x\_offset, numericStandard y\_offset, numericStandard z\_offset)

### Public Attributes

- numericStandard `rotation` [3][3]
- numericStandard `translation` [3]
- long `timestamp`
- long `frameNumber`

### 5.23.1 Detailed Description

```
template<typename numericStandard>struct transformComponent2< numericStandard >
```

Contains a Frame Transformation with a 3x3 rotation matrix.

## 5.23.2 Member Function Documentation

5.23.2.1 `template<typename numericStandard> void transformComponent2< numericStandard >::setTransform ( numericStandard x, numericStandard y, numericStandard z, numericStandard input_rotation[3][3], long timestamp ) [inline]`

Initialize the frame transformation with the translation vector {x,y,z} and the rotation matrix input\_rotation, and the timestamp.

## 5.23.3 Member Data Documentation

5.23.3.1 `template<typename numericStandard> long transformComponent2< numericStandard >::frameNumber`

The frame number of the sensor.

5.23.3.2 `template<typename numericStandard> numericStandard transformComponent2< numericStandard >::rotation[3][3]`

Rotation matrix of the frame transformation

5.23.3.3 `template<typename numericStandard> long transformComponent2< numericStandard >::timestamp`

unix timestamp in milliseconds

5.23.3.4 `template<typename numericStandard> numericStandard transformComponent2< numericStandard >::translation[3]`

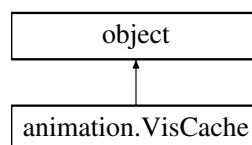
translation vector of the frame transformation

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

- include/typedef.h

## 5.24 animation.VisCache Class Reference

Inheritance diagram for animation.VisCache:



### Static Public Attributes

- dictionary **cache** = {}

### 5.24.1 Detailed Description

Provides global references to VTK actors in the visualization.

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

- vis/animation.py

## 5.25 animation.vtkScissorCallback Class Reference

### Public Member Functions

- def `__init__` (self, data\_source, renderer)
- def `execute` (self, obj, event)

### Public Attributes

- `timer_count`
- `data_source`
- `render_window`
- `renderer`

#### 5.25.1 Detailed Description

VTK Timer Callback that modifies the Scissor Actor

#### 5.25.2 Member Function Documentation

##### 5.25.2.1 def animation.vtkScissorCallback.execute ( self, obj, event )

if more than SCISSOR\_REFRESH\_RATE\_MS milliseconds has elapsed since the last scissor rendering (DataCache.cache['last\_scissor\_update']), and the scissor pose has been updated (DataCache.cache['scissor\_update']), then the scissor's pose will be adjusted.

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

- vis/animation.py

## 5.26 animation.vtkTimerCallback Class Reference

### Public Member Functions

- def `__init__` (self, data\_source, renderer)
- def `execute` (self, obj, event)

### Public Attributes

- `timer_count`
- `data_source`
- `render_window`
- `renderer`

### 5.26.1 Detailed Description

General VTK Timer Callback function.

### 5.26.2 Constructor & Destructor Documentation

#### 5.26.2.1 `def animation.vtkTimerCallback.__init__( self, data_source, renderer )`

Arguments

-----

```
data_source : str
    enum : 'line_of_cut', 'plane_of_cut'
    specifies which callback this is
renderer : vtk.Renderer
```

### 5.26.3 Member Function Documentation

#### 5.26.3.1 `def animation.vtkTimerCallback.execute( self, obj, event )`

VTK Timer Callback function

If the `line_of_cut` or `plane_of_cut` data has not been updated or the `line_of_cut` or `plane_of_cut` has been rendered

If the VTK actors are modified, then `render_window.Render()` is called

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

- `vis/animation.py`

# Index

- `__init__`
    - `animation::vtkTimerCallback`, 36
  - `addControlButton`
    - `DataCollectionGroupBox`, 12
  - `addCutTrainingData`
    - `scissorTraining`, 28
  - `addInitialCloudData`
    - `scissorTraining`, 28
  - `addLineData`
    - `scissorTraining`, 28
  - `addNasalPoints`
    - `SeptumSurfaceGenerator`, 30
  - `addRadioButton`
    - `DataCollectionGroupBox`, 12
  - `addSeptumPoints`
    - `SeptumSurfaceGenerator`, 30
  - `animation`, 7
    - `create_cube_actor`, 8
    - `create_training_line`, 8
    - `draw_convex_hull`, 8
    - `get_points_for_cube_from_plane`, 8
    - `load_data`, 9
    - `main`, 9
    - `mkVtkIdList`, 9
    - `PACKET_PREFIXES`, 9
    - `process_line`, 9
    - `process_stdin`, 9
  - `animation.DataCache`, 11
  - `animation.VisCache`, 34
  - `animation.vtkScissorCallback`, 35
  - `animation.vtkTimerCallback`, 35
  - `animation::vtkScissorCallback`
    - `execute`, 35
  - `animation::vtkTimerCallback`
    - `__init__`, 36
    - `execute`, 36
- `bladeLength`
  - `scissorModel`, 26
- `bufferedReader`, 10
- `bufferedReader.NonBlockingNewlineBufferedReader`, 19
- `bufferedReader.TestNonBlockingNewlineBufferedReader`
  - `Reader`, 31
- `call_phantom_add_points`
  - `Platform`, 22
- `call_scissorTrain_add_cut_training_data`
  - `Platform`, 22
- `call_scissorTrain_add_initial_cloud`
  - `Platform`, 22
- `call_scissorTrain_add_line`
  - `Platform`, 22
- `call_scissorTrain_add_scissor_normal_trace_data`
  - `Platform`, 22
- `call_scissorTrain_training_solve`
  - `Platform`, 22
- `call_septumEstimate_add_nasal_points`
  - `Platform`, 23
- `call_septumEstimate_add_septum_points`
  - `Platform`, 23
- `call_septumEstimate_generate_plane`
  - `Platform`, 23
- `callScissorIncrementActualCutIndex`
  - `Platform`, 23
- `checkedButton`
  - `DataCollectionGroupBox`, 12
- `clearCache`
  - `Platform`, 23
- `clearMode`
  - `Platform`, 23
- `clickCalibrationDataRecordGroupBox`
  - `DataCollectionWidget`, 13
- `clickOperationalDataRecordGroupBox`
  - `DataCollectionWidget`, 13
- `coordinate`, 11
- `corner_defined`
  - `planeFromLineCoordinate`, 21
- `cornerPoints`
  - `planeFromLineCoordinate`, 21
- `create_cube_actor`
  - `animation`, 8
- `create_training_line`
  - `animation`, 8
- `cut_directions`
  - `trainingData`, 32
- `DataCollectionGroupBox`, 12
  - `addControlButton`, 12
  - `addRadioButton`, 12
  - `checkedButton`, 12
  - `DataCollectionGroupBox`, 12
- `DataCollectionWidget`, 13
  - `clickCalibrationDataRecordGroupBox`, 13
  - `clickOperationalDataRecordGroupBox`, 13
  - `releaseCalibrationCalculationButton`, 13
  - `releaseCalibrationDataRecordGroupBox`, 14
  - `releaseOperationalCalculationButton`, 14
  - `releaseOperationalDataRecordGroupBox`, 14
  - `setModeHandler`, 14

- DataReceiveMode
  - Platform, 22
- dataReceiveMode
  - Platform, 25
- DataRecordButton, 14
- DataRecordGroupBox, 15
  - DataRecordGroupBox, 15
  - getCalculationButton, 15
  - getStartStopRecordButton, 15
- draw\_convex\_hull
  - animation, 8
- dumpTrainingData
  - scissorTraining, 28
- endPoint
  - lineCoordinate, 17
- eulerXYZ
  - lineCoordinate, 17
- execute
  - animation::vtkScissorCallback, 35
  - animation::vtkTimerCallback, 36
- feedback, 10
- feedback::Properties, 25
  - getScissorModelFile, 25
  - getSensorSerial, 25
  - getTrainingDataFile, 25
  - Properties, 25
- FeedbackApplication, 15
  - FeedbackApplication, 16
- frameNumber
  - transformComponent, 33
  - transformComponent2, 34
- generatePhantomPlane
  - SeptumSurfaceGenerator, 30
- generatePointCloudFromVector
  - SeptumSurfaceGenerator, 31
- generateSeptumPlane
  - SeptumSurfaceGenerator, 31
- get\_points\_for\_cube\_from\_plane
  - animation, 8
- getAverageInitialPoint
  - scissorTraining, 28
- getCalculationButton
  - DataRecordGroupBox, 15
- getLineDirection
  - scissorTraining, 28
- getScissorModelFile
  - feedback::Properties, 25
- getSensorSerial
  - feedback::Properties, 25
- getStartStopRecordButton
  - DataRecordGroupBox, 15
- getTrainingDataFile
  - feedback::Properties, 25
- handleCalibrationCalculation
  - ModeHandler, 19
- handleModeChange
  - ModeHandler, 19
- handleOperationalCalculation
  - ModeHandler, 19
- haveNasalPoints
  - SeptumSurfaceGenerator, 31
- incrementActualCutIndex
  - scissorTraining, 28
- isDataValidForReceiveMode
  - Platform, 23
- lineCoordinate, 16
  - endPoint, 17
  - eulerXYZ, 17
  - print, 16
  - scissorPinchPoint, 17
  - startPoint, 17
  - timestamp, 17
  - valid, 17
- lineOfCutGenerator, 17
- load\_data
  - animation, 9
- loadScissorTrainingData
  - Platform, 23
- main
  - animation, 9
- MainWindow, 18
- mainloop
  - Platform, 23
- mkVtkIdList
  - animation, 9
- ModeHandler, 18
  - handleCalibrationCalculation, 19
  - handleModeChange, 19
  - handleOperationalCalculation, 19
  - ModeHandler, 19
- n\_blade\_points
  - scissorModel, 26
- normalVector
  - scissorModel, 26
- openScissorModel
  - Platform, 24
- PACKET\_PREFIXES
  - animation, 9
- pivotCalibratePinchPoint
  - scissorTraining, 28
- plane\_normal
  - trainingData, 32
- planeFromLineCoordinate, 20
  - corner\_defined, 21
  - cornerPoints, 21
  - planeFromLineCoordinate, 20
  - print, 21
- Platform, 21
  - call\_phantom\_add\_points, 22



- call\_scissorTrain\_add\_cut\_training\_data, 22
- call\_scissorTrain\_add\_initial\_cloud, 22
- call\_scissorTrain\_add\_line, 22
- call\_scissorTrain\_add\_scissor\_normal\_trace\_data, 22
- call\_scissorTrain\_training\_solve, 22
- call\_septumEstimate\_add\_nasal\_points, 23
- call\_septumEstimate\_add\_septum\_points, 23
- call\_septumEstimate\_generate\_plane, 23
- callScissorIncrementActualCutIndex, 23
- clearCache, 23
- clearMode, 23
- DataReceiveMode, 22
- dataReceiveMode, 25
- isDataValidForReceiveMode, 23
- loadScissorTrainingData, 23
- mainloop, 23
- openScissorModel, 24
- receiveMessage, 24
- saveScissorModel, 24
- saveScissorTrainingData, 24
- setCacheMode, 24
- setDataReceiveMode, 24
- setLineOfCutMode, 24
- setScissorValidateMode, 24
- setToolCutValidationMode, 24
- print
  - lineCoordinate, 16
  - planeFromLineCoordinate, 21
  - scissorModel, 26
- printNewTrainingLineData
  - scissorTraining, 28
- printTrainingLineDataAtIndex
  - scissorTraining, 28
- process\_line
  - animation, 9
- process\_stdin
  - animation, 9
- Properties
  - feedback::Properties, 25
- quaternion
  - transformComponent, 33
- receiveMessage
  - Platform, 24
- releaseCalibrationCalculationButton
  - DataCollectionWidget, 13
- releaseCalibrationDataRecordGroupBox
  - DataCollectionWidget, 14
- releaseOperationalCalculationButton
  - DataCollectionWidget, 14
- releaseOperationalDataRecordGroupBox
  - DataCollectionWidget, 14
- rotation
  - transformComponent2, 34
- saveScissorModel
  - Platform, 24
- saveScissorTrainingData
  - Platform, 24
- scissor\_model\_id
  - scissorModel, 26
- scissorModel, 26
  - bladeLength, 26
  - n\_blade\_points, 26
  - normalVector, 26
  - print, 26
  - scissor\_model\_id, 26
  - sensorTranslationToPinchPoint, 26
  - valid, 26
- scissorPinchPoint
  - lineCoordinate, 17
- scissorTraining, 27
  - addCutTrainingData, 28
  - addInitialCloudData, 28
  - addLineData, 28
  - dumpTrainingData, 28
  - getAverageInitialPoint, 28
  - getLineDirection, 28
  - incrementActualCutIndex, 28
  - pivotCalibratePinchPoint, 28
  - printNewTrainingLineData, 28
  - printTrainingLineDataAtIndex, 28
  - scissorTraining, 27
  - setDebugMode, 28
  - setPlaneNormals, 29
  - setScissorModelForValidation, 29
  - setScissorNormal, 29
  - setTrainingData, 29
  - solveScissorParameters, 29
  - validate, 29
  - validateWithActualCut, 29
- sensorTranslationToPinchPoint
  - scissorModel, 26
- septumSurface, 30
- SeptumSurfaceGenerator, 30
  - addNasalPoints, 30
  - addSeptumPoints, 30
  - generatePhantomPlane, 30
  - generatePointCloudFromVector, 31
  - generateSeptumPlane, 31
  - haveNasalPoints, 31
  - setDistanceThreshold, 31
- setCacheMode
  - Platform, 24
- setDataReceiveMode
  - Platform, 24
- setDebugMode
  - scissorTraining, 28
- setDistanceThreshold
  - SeptumSurfaceGenerator, 31
- setLineOfCutMode
  - Platform, 24
- setModeHandler
  - DataCollectionWidget, 14
- setPlaneNormals

- scissorTraining, 29
- setScissorModelForValidation
  - scissorTraining, 29
- setScissorNormal
  - scissorTraining, 29
- setScissorValidateMode
  - Platform, 24
- setToolCutValidationMode
  - Platform, 24
- setTrainingData
  - scissorTraining, 29
- setTransform
  - transformComponent2, 34
- solveScissorParameters
  - scissorTraining, 29
- startPoint
  - lineCoordinate, 17
- starting\_points
  - trainingData, 32
- timestamp
  - lineCoordinate, 17
  - transformComponent, 33
  - transformComponent2, 34
- trainingData, 32
  - cut\_directions, 32
  - plane\_normal, 32
  - starting\_points, 32
- transformComponent
  - frameNumber, 33
  - quaternion, 33
  - timestamp, 33
  - translation, 33
- transformComponent< numericStandard >, 32
- transformComponent2
  - frameNumber, 34
  - rotation, 34
  - setTransform, 34
  - timestamp, 34
  - translation, 34
- transformComponent2< numericStandard >, 33
- translation
  - transformComponent, 33
  - transformComponent2, 34
- valid
  - lineCoordinate, 17
  - scissorModel, 26
- validate
  - scissorTraining, 29
- validateWithActualCut
  - scissorTraining, 29