

LIDAR Controller

v1.0

Generated by Doxygen 1.8.11

Contents

1	Namespace Index	1
1.1	Packages	1
2	Hierarchical Index	3
2.1	Class Hierarchy	3
3	Class Index	5
3.1	Class List	5
4	File Index	7
4.1	File List	7
5	Namespace Documentation	9
5.1	LIDAR_Controller Namespace Reference	9
6	Class Documentation	11
6.1	LIDAR_Controller.DAO Class Reference	11
6.1.1	Detailed Description	11
6.1.2	Member Function Documentation	12
6.1.2.1	loadMeasurements()	12
6.1.2.2	saveMeasurements(List< Measurement > l)	13
6.2	LIDAR_Controller.MainWindow Class Reference	14
6.2.1	Detailed Description	15
6.2.2	Constructor & Destructor Documentation	16
6.2.2.1	MainWindow()	16
6.2.3	Member Function Documentation	16

6.2.3.1	button2_Click(object sender, RoutedEventArgs e)	16
6.2.3.2	button3_Click(object sender, RoutedEventArgs e)	17
6.2.3.3	button6_Click(object sender, RoutedEventArgs e)	17
6.2.3.4	comboBox2_SelectionChanged(object Sender, SelectionChangedEventArgs e)	18
6.2.3.5	ComPortReceiveHandler(object Sender, SerialDataReceivedEventArgs e)	18
6.2.3.6	enableVisuals(bool b)	19
6.2.3.7	entfMessung_Click(object sender, RoutedEventArgs e)	20
6.2.3.8	ExceptionHandler(Exception ex)	21
6.2.3.9	Init()	22
6.2.3.10	Init3D()	22
6.2.3.11	loadBtn_Click(object sender, RoutedEventArgs e)	23
6.2.3.12	neuMessung_Click(object sender, RoutedEventArgs e)	24
6.2.3.13	offsetBtn_Click(object sender, RoutedEventArgs e)	25
6.2.3.14	posBtn_Click(object sender, RoutedEventArgs e)	26
6.2.3.15	reverse_Click(object sender, RoutedEventArgs e)	26
6.2.3.16	saveBtn_Click(object sender, RoutedEventArgs e)	27
6.2.3.17	sendBtn_Click(object sender, RoutedEventArgs e)	27
6.2.3.18	verbinden_Click(object sender, RoutedEventArgs e)	28
6.3	LIDAR_Controller.Measurement Class Reference	29
6.3.1	Detailed Description	31
6.3.2	Constructor & Destructor Documentation	31
6.3.2.1	Measurement()	31
6.3.2.2	Measurement(Vector3D linearOffset, double rotaryOffsetX, double rotaryOffsetZ)	31
6.3.3	Member Function Documentation	32
6.3.3.1	Equals(Measurement m)	32
6.3.3.2	getDistanceData(int mpos, int spos)	33
6.3.3.3	getGeometry3D()	33
6.3.3.4	makeGeometry3D()	34
6.3.3.5	Refresh()	35
6.3.3.6	setDistanceData(int mpos, int spos, int data)	35
6.3.3.7	setGeometryPoint3D(int i, int k)	36
6.3.3.8	Turn3DVektorXZ(Vector3D v, double mdegree, double sdegree)	37
6.3.4	Property Documentation	37
6.3.4.1	id	37
6.3.4.2	linearOffset	38
6.3.4.3	mId	38
6.3.4.4	rotaryOffsetX	38
6.3.4.5	rotaryOffsetZ	38

7 File Documentation	39
7.1 LIDAR_VIS_TEST/LIDAR_WPF_TEST/DAO.cs File Reference	39
7.1.1 Detailed Description	39
7.2 LIDAR_VIS_TEST/LIDAR_WPF_TEST/MainWindow.xaml.cs File Reference	39
7.2.1 Detailed Description	39
7.3 LIDAR_VIS_TEST/LIDAR_WPF_TEST/Measurement.cs File Reference	40
7.3.1 Detailed Description	40
Index	41

Chapter 1

Namespace Index

1.1 Packages

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

LIDAR_Controller	9
--	---

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

LIDAR_Controller.DAO	11
LIDAR_Controller.Measurement	29
Window	
LIDAR_Controller.MainWindow	14

Chapter 3

Class Index

3.1 Class List

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

LIDAR_Controller.DAO	
A "data access object" class. This is used to separate the file operations from the GUI logic . .	11
LIDAR_Controller.MainWindow	
The application's main form. This class contains all necessary event and exception handlers .	14
LIDAR_Controller.Measurement	
(Serializable) a measurement	29

Chapter 4

File Index

4.1 File List

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

LIDAR_VIS_TEST/LIDAR_WPF_TEST/ DAO.cs	
Implements the dao class	39
LIDAR_VIS_TEST/LIDAR_WPF_TEST/ MainWindow.xaml.cs	
Implements the main window.xaml class. This file contains all GUI specific code (Event handler for GUI-elements)	39
LIDAR_VIS_TEST/LIDAR_WPF_TEST/ Measurement.cs	
Implements the measurement class	40

Chapter 5

Namespace Documentation

5.1 LIDAR_Controller Namespace Reference

Classes

- class [DAO](#)
A "data access object" class. This is used to separate the file operations from the GUI logic.
- class [MainWindow](#)
The application's main form. This class contains all necessary event and exception handlers.
- class [Measurement](#)
(Serializable) a measurement.

Chapter 6

Class Documentation

6.1 LIDAR_Controller.DAO Class Reference

A "data access object" class. This is used to separate the file operations from the GUI logic.

Static Public Member Functions

- static List< [Measurement](#) > [loadMeasurements](#) ()
Loads the measurements from a xml file. First ssk user if he/she wants to load data. If not return null.
- static void [saveMeasurements](#) (List< [Measurement](#) > l)
Saves the measurements to a xml file.

6.1.1 Detailed Description

A "data access object" class. This is used to separate the file operations from the GUI logic.

Author

Alexander Miller (7089316)

Date

22.12.2015

Definition at line 31 of file DAO.cs.

6.1.2 Member Function Documentation

6.1.2.1 `public static List< Measurement > LIDAR_Controller.DAO.loadMeasurements () [static]`

Loads the measurements from a xml file. First ask user if he/she wants to load data. If not return null.

a: If no problems occur then...

1. Open a file selector
2. If a file was selected, open it
3. Create a XmlSerializer
4. Deserialize the data into a list
5. Get the maximum measurement id and set it
6. Close the file
7. Inform user about successfully loading all data

b: If something goes wrong, show a MessageBox.

Author

Alexander Miller (7089316)

Date

22.12.2015

Returns

A list of all measurements.

Definition at line 57 of file DAO.cs.

```

58         {
59             if (MessageBox.Show("Möchten Sie Messergebnisse aus einer Datei laden?\nDabei gehen nicht
gespeicherte Messungen verloren!", "Warnung!", MessageBoxButtons.OKCancel, MessageBoxIcon.Warning) ==
MessageBoxResult.OK)
60             {
61                 List<Measurement> list;
62                 //a.
63                 try
64                 {
65                     //1.
66                     OpenFileDialog ofdialog = new OpenFileDialog();
67                     ofdialog.DefaultExt = ".lmd"; //LIDAR Measurement Data
68                     ofdialog.Filter = "LIDAR Measurement Data|*.lmd";
69                     Nullable<bool> r = ofdialog.ShowDialog();
70                     if (r == true)
71                     {
72                         //2.
73                         FileStream fs = new FileStream(ofdialog.FileName, FileMode.Open);
74                         //3.
75                         Type t = typeof(List<Measurement>);
76                         XmlSerializer serializer = new XmlSerializer(t);
77                         //4.
78                         list = (List<Measurement>)serializer.Deserialize(fs);
79                         //5.
80                         Measurement.id = list[list.Count-1].mId+1;
81                         //6.
82                         fs.Close();
83                         //7.
84                         MessageBox.Show("Laden erfolgreich.", "Info", MessageBoxButtons.OK, MessageBoxIcon.
Information);
85                         return list;
86                     }
87                 }
88             }
89             //b.
90             catch (Exception E)
91             {
92                 MessageBox.Show(E.Message, "Fehler!", MessageBoxButtons.OK, MessageBoxIcon.Error);
93             }
94         }
95         return null;
96     }

```

6.1.2.2 public static void LIDAR_Controller.DAO.saveMeasurements (List< Measurement > /) [static]

Saves the measurements to a xml file.

a. If no problems occur then...

1. Open a file selector
2. If a file was selected, create it
3. Create a XmlSerializer
4. Serialize the data into the file
5. Close the file
6. Inform user about successfully saving all data

b. If something goes wrong, show a MessageBox.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

/	The List< Measurement > to process.
---	---

Definition at line 120 of file DAO.cs.

```

121     {
122         //a.
123         try
124         {
125             //1.
126             SaveFileDialog sfdialog = new SaveFileDialog();
127             sfdialog.DefaultExt = ".lmd"; //LIDAR Measurment Data
128             sfdialog.Filter = "LIDAR Measurment Data|*.lmd";
129             Nullable<bool> r = sfdialog.ShowDialog();
130             if (r == true)
131             {
132                 XmlWriterSettings xmlWriterSettings = new XmlWriterSettings();
133                 xmlWriterSettings.Indent = true;
134                 //2.
135                 FileStream fs = new FileStream(sfdialog.FileName, FileMode.Create);
136                 //3.
137                 Type t = typeof(List<Measurement>);
138                 XmlSerializer serializer = new XmlSerializer(t);
139                 //4.
140                 XmlWriter xmlWriter = XmlWriter.Create(fs, xmlWriterSettings);
141                 serializer.Serialize(xmlWriter, l);
142                 //5.
143                 fs.Close();
144                 //6.
145                 MessageBox.Show("Speichern erfolgreich.", "Info", MessageBoxButton.OK, MessageBoxImage.
Information);
146             }
147         }
148         //b.
149         catch (Exception E)
150         {
151             MessageBox.Show(E.Message, "Fehler!", MessageBoxButton.OK, MessageBoxImage.Error);
152         }
153     }
154 }
```

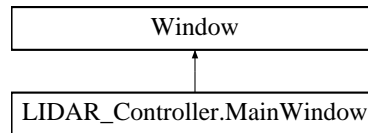
The documentation for this class was generated from the following file:

- LIDAR_VIS_TEST/LIDAR_WPF_TEST/[DAO.cs](#)

6.2 LIDAR_Controller.MainWindow Class Reference

The application's main form. This class contains all necessary event and exception handlers.

Inheritance diagram for LIDAR_Controller.MainWindow:



Public Member Functions

- [MainWindow](#) ()

Default constructor of [MainWindow](#). This constructor does all the initialisation work. At first it initialises all GUI objects. Then it calls the `init()` function. This function adds COM-Port informations to some GUI objects. The last function initializes the Viewport object.

Private Member Functions

- void [ComPortReceiveHandler](#) (object Sender, SerialDataReceivedEventArgs e)
Handler, called when the COM-port receives something.
- void [ExeptionHandler](#) (Exception ex) \D+ "
Handler, called when a exception occurs.
- void [Init](#) ()
Initialises both combo boxes with relevant data (e.g. available COM-Ports).
- void [comboBox2_SelectionChanged](#) (object Sender, SelectionChangedEventArgs e)
Event handler. Called by comboBox2 for selection changed events. This handler gets the offset values of the selected measurement and displays them.
- void [Init3D](#) ()
Initialises the viewport object.
- void [verbinden_Click](#) (object sender, RoutedEventArgs e)
Event handler. Called by verbinden for click events.
- void [button2_Click](#) (object sender, RoutedEventArgs e)
Event handler. Called by button2 for click events. This functions sends the "#1" (Onetime Measure) command to the LIDAR-Scanner.
- void [button3_Click](#) (object sender, RoutedEventArgs e)
Event handler. Called by button3 for click events. This functions sends the "#2" (Radar mode 2D) command to the LIDAR-Scanner.
- void [posBtn_Click](#) (object sender, RoutedEventArgs e)
Event handler. Called by posBtn for click events. This functions sends the "#3" (Set Position) command with the position infos from "txt_MPos" & "txt_SPos" to the LIDAR-Scanner.
- void [button6_Click](#) (object sender, RoutedEventArgs e)
Event handler. Called by button6 for click events. This functions sends the "#4" (Calibration) command to the LIDAR-Scanner.
- void [neuMessung_Click](#) (object sender, RoutedEventArgs e)
Event handler. Called by neuMessung for click events. This function adds a new [Measurement](#) to MeasureList and refreshes comboBox2.
- void [entfMessung_Click](#) (object sender, RoutedEventArgs e)

Event handler. Called by entfMessung for click events. This functions deletes the selected measurement and updates combobox2.

- void [offsetBtn_Click](#) (object sender, RoutedEventArgs e)

Event handler. Called by offsetBtn for click events. This function gets the offset information. Then it adds this values to the selected measurement.

- void [saveBtn_Click](#) (object sender, RoutedEventArgs e)

Event handler. Called by saveBtn for click events. Calls "DAO.saveMeasurements" to save all measurements to a xml file.

- void [loadBtn_Click](#) (object sender, RoutedEventArgs e)

Event handler. Called by loadBtn for click events. Calls "DAO.loadMeasurements" to load measurements from a xml file.

- void [sendBtn_Click](#) (object sender, RoutedEventArgs e)

Event handler. Called by sendBtn for click events. This function sends the text of "sendTxt" to the LIDAR-Scanner.

- void [enableVisuals](#) (bool b)

Enables/Disables the visuals.

- void [reverse_Click](#) (object sender, RoutedEventArgs e)

Event handler. Called by reverse for click events. This function transforms the 3D model to a open/closed shape.

Private Attributes

- SerialPort [ComPort](#) = null

The COM port which is used to communicate with the LIDAR-Scanner.

- int [selectedMeasure](#)

The actually selected measurement.

- List< [Measurement](#) > [MeasureList](#) = new List<[Measurement](#)>()

List of all measurements.

- HelixViewport3D [myViewport](#) = new HelixViewport3D()

A viewport object that handles all 3D-Drawings.

- GridLinesVisual3D [gridLinesXY](#) = new GridLinesVisual3D()

The grid lines of the xy-plane.

6.2.1 Detailed Description

The application's main form. This class contains all necessary event and exception handlers.

Author

Alexander Miller (7089316)

Date

22.12.2015

Definition at line 31 of file MainWindow.xaml.cs.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 public LIDAR_Controller.MainWindow.MainWindow ()

Default constructor of [MainWindow](#). This constructor does all the initialisation work. At first it initialises all GUI objects. Then it calls the `init()` function. This function adds COM-Port informations to some GUI objects. The last function initializes the Viewport object.

Author

Alexander Miller (7089316)

Date

22.12.2015

Definition at line 65 of file MainWindow.xaml.cs.

```

66         {
67             InitializeComponent();
68             Init();
69             Init3D();
70         }

```

6.2.3 Member Function Documentation

6.2.3.1 private void LIDAR_Controller.MainWindow.button2_Click (object sender, RoutedEventArgs e) [private]

Event handler. Called by button2 for click events. This functions sends the "#1" (Onetime Measure) command to the LIDAR-Scanner.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>sender</i>	Source of the event.
<i>e</i>	Routed event information.

Definition at line 361 of file MainWindow.xaml.cs.

```

362         {
363             ComPort.WriteLine("#1");
364         }

```

6.2.3.2 private void LIDAR_Controller.MainWindow.button3_Click (object *sender*, RoutedEventArgs *e*) [private]

Event handler. Called by button3 for click events. This functions sends the "#2" (Radar mode 2D) command to the LIDAR-Scanner.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>sender</i>	Source of the event.
<i>e</i>	Routed event information.

Definition at line 379 of file MainWindow.xaml.cs.

```
380         {  
381             ComPort.WriteLine("#2");  
382         }
```

6.2.3.3 private void LIDAR_Controller.MainWindow.button6_Click (object *sender*, RoutedEventArgs *e*) [private]

Event handler. Called by button6 for click events. This functions sends the "#4" (Calibration) command to the LIDAR-Scanner.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>sender</i>	Source of the event.
<i>e</i>	Routed event information.

Definition at line 427 of file MainWindow.xaml.cs.

```
428         {  
429             ComPort.WriteLine("#4");  
430         }
```

6.2.3.4 void LIDAR_Controller.MainWindow.comboBox2_SelectionChanged (object *Sender*, SelectionChangedEventArgs *e*) [private]

Event handler. Called by comboBox2 for selection changed events. This handler gets the offset values of the selected measurement and displays them.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>Sender</i>	Source of the event.
<i>e</i>	Selection changed event information.

Definition at line 206 of file MainWindow.xaml.cs.

```

207         {
208             selectedMeasure = comboBox2.SelectedIndex;
209             if (selectedMeasure >= 0)
210             {
211                 xOffsetTxt.Text = "" + MeasureList[selectedMeasure].linearOffset.
X;
212                 yOffsetTxt.Text = "" + MeasureList[selectedMeasure].linearOffset.
Y;
213                 zOffsetTxt.Text = "" + MeasureList[selectedMeasure].linearOffset.
Z;
214                 degOffsetXTxt.Text = "" + MeasureList[selectedMeasure].
rotaryOffsetX;
215                 degOffsetZTxt.Text = "" + MeasureList[selectedMeasure].
rotaryOffsetZ;
216             }
217         }
218     }

```

6.2.3.5 private void LIDAR_Controller.MainWindow.ComPortReceiveHandler (object *Sender*, SerialDataReceivedEventArgs *e*) [private]

Handler, called when the COM-port receives something.

a: if no problems occur then...

1. Check if ComPort is open
2. Read line from buffer
3. Add the received line to textBox1
4. Get relevant data from string using a regular expression
5. Parse data into integers
6. Set a point of the selected measurement based on the received data.

b: If something goes wrong, call the exception handler.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>Sender</i>	Source of the event.
<i>e</i>	Serial data received event information.

Definition at line 95 of file MainWindow.xaml.cs.

```

96         {
97             //a
98             try
99             {
100                 //1
101                 if (!ComPort.IsOpen) return;
102                 string ReceivedText = "";
103                 //2
104                 ReceivedText = ComPort.ReadLine();
105                 //3
106                 Dispatcher.BeginInvoke(new Action(() =>
107                 {
108                     this.textBox1.AppendText(ReceivedText);
109                     this.textBox1.ScrollToEnd();
110                 }));
111                 //4
112                 string[] numbers = Regex.Split(ReceivedText, @"\D+");
113                 //5
114                 if (numbers.Length == 5)
115                 {
116                     int mpos = 0;
117                     int spos = 0;
118                     int value = 0;
119                     if (int.TryParse(numbers[1], out mpos) && int.TryParse(numbers[2], out spos) && int.
120 TryParse(numbers[3], out value))
121                     //6
122                     MeasureList[selectedMeasure].setDistanceData(mpos+1, spos
123 , value);
124                     Dispatcher.BeginInvoke(new Action(() =>
125                     {
126                         MeasureList[selectedMeasure].setGeometryPoint3D(mpos+1 ,
127 spos);
128                     }));
129                 }
130             }
131         }
132         //b
133         catch (Exception ex) { ExeptionHandler(ex); }
134     }

```

6.2.3.6 private void LIDAR_Controller.MainWindow.enableVisuals (bool b) [private]

Enables/Disables the visuals.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>b</i>	true enables visuals and false disables them.
----------	---

Definition at line 654 of file MainWindow.xaml.cs.

```

655     {
656         //neuMessung.IsEnabled = b;
657         //entfMessung.IsEnabled = b;
658         //offsetBtn.IsEnabled = b;
659         btn_einzelmessung.IsEnabled = b;
660         btn_kalibrieren.IsEnabled = b;
661         posBtn.IsEnabled = b;
662         btn_radar.IsEnabled = b;
663         txt_MPos.IsEnabled = b;
664         txt_SPos.IsEnabled = b;
665         //sendBtn.IsEnabled = b;
666         //comboBox2.IsEnabled = b;
667         //sendTxt.IsEnabled = b;
668         //txt_punktmessung.IsEnabled = b;
669         //xOffsetTxt.IsEnabled = b;
670         //yOffsetTxt.IsEnabled = b;
671         //zOffsetTxt.IsEnabled = b;
672         //degOffsetTxt.IsEnabled = b;
673         //textBox1.IsEnabled = b;
674     }

```

6.2.3.7 private void LIDAR_Controller.MainWindow.entfMessung_Click (object sender, RoutedEventArgs e) [private]

Event handler. Called by entfMessung for click events. This functions deletes the selected measurement and updates comboBox2.

1. Check if at least 2 measurements are available
2. Remove the selected measurement form MeasureList
3. Clear comboBox2
4. Add all remaining measurements to comboBox2
5. Reset myViewport
6. Add all remaining measurements to myViewport as children to display them

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>sender</i>	Source of the event.
<i>e</i>	Routed event information.

Definition at line 486 of file MainWindow.xaml.cs.

```

487     {
488         //1
489         if (MeasureList.Count > 1)
490         {
491             //2
492             MeasureList.RemoveAt(selectedMeasure);
493             //3
494             comboBox2.Items.Clear();
495             //4
496             foreach (Measurement x in MeasureList)
497             {
498                 comboBox2.Items.Add(x);
499             }
500             comboBox2.SelectedIndex = 0;
501             //5
502             myViewport.Children.Clear();
503             //6
504             myViewport.Children.Add(gridLinesXY);
505             myViewport.Children.Add(new DefaultLights());
506             foreach (Measurement x in MeasureList)
507             {
508                 myViewport.Children.Add(x.getGeometry3D());
509             }
510         }
511     }
512 }
513 }
```

6.2.3.8 private void LIDAR_Controller.MainWindow.ExceptionHandler (Exception ex)\D+ [private]

Handler, called when a exception occurs.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

ex	The exception that occurred.
----	------------------------------

Definition at line 147 of file MainWindow.xaml.cs.

```

148     {
149         switch (ex.GetType().Name)
150         {
151             case "IOException":
152                 MessageBox.Show("Name: " + ex.GetType().Name + "\r\nBeschreibung:\r\n" + ex.Message, "
Fehler!", MessageBoxButton.OK, MessageBoxImage.Error);
153                 break;
154             case "ArgumentException":
155                 MessageBox.Show("Bitte Eingaben überprüfen!", "Fehleingabe erkannt!", MessageBoxButton.
OK, MessageBoxImage.Information);
156                 break;
157             default:
158                 MessageBox.Show("Name: " + ex.GetType().Name + "\r\nBeschreibung:\r\n" + ex.Message, "
Unbekannter Fehler!", MessageBoxButton.OK, MessageBoxImage.Error);
159                 break;
160         }
161     }
162 }
```

6.2.3.9 private void LIDAR_Controller.MainWindow.Init () [private]

Initialises both combo boxes with relevant data (e.g. available COM-Ports).

Author

Alexander Miller (7089316)

Date

22.12.2015

Definition at line 173 of file MainWindow.xaml.cs.

```
174     {
175         //add all COM-Ports to combobox
176         foreach (string ports in SerialPort.GetPortNames())
177         {
178             comboBox.Items.Add(ports);
179         }
180         //initial measurement
181         MeasureList.Add(new Measurement(new Vector3D(0, 0, 0.5), 0, 0));
182
183         //add measurement to combobox2 and register a new event handler for "SelectionChangedEvent"
184         comboBox2.Items.Add(MeasureList[selectedMeasure]);
185         comboBox2.DisplayMemberPath = "mId";
186         comboBox2.SelectionChanged += new SelectionChangedEventHandler(
187             comboBox2_SelectionChanged);
188         comboBox2.SelectedIndex = 0;
189         //Enable/Disable all GUI-Elements that need a connected COM-Port
190         enableVisuals(false);
191     }
```

6.2.3.10 private void LIDAR_Controller.MainWindow.Init3D () [private]

Initialises the viewport object.

1. Configure the grid lines for the xy-Plane.
2. Configure myViewport to show additional data.
3. Add gridLinesXY, a default light and the 3D-Model of the selected measurement to myViewport as children.
4. Add myViewport to grid (GUI-Element) as child and refresh.

Author

Alexander Miller (7089316)

Date

22.12.2015

Definition at line 233 of file MainWindow.xaml.cs.

```

234     {
235         //1
236         gridLinesXY.MinorDistance = 10; //Distance in cm for the minor lines
237         gridLinesXY.MajorDistance = 100; //Distance in cm for the major lines
238         gridLinesXY.Thickness = 1.5; //Line thickness
239         gridLinesXY.Length = 10000; //Maximum grid size
240         gridLinesXY.Width = 10000; //Maximum grid size
241
242         //2
243         myViewport.ShowFrameRate = true; //Show framerate
244         myViewport.ShowCoordinateSystem = true; //Show small Coordinate system
245         myViewport.ShowFieldOfView = true; //Show Field of View
246
247         //3
248         myViewport.Children.Add(gridLinesXY);
249         myViewport.Children.Add(new DefaultLights());
250         myViewport.Children.Add(MeasureList[
selectedMeasure].getGeometry3D());
251
252         //4
253         grid.Children.Add(myViewport);
254         grid.UpdateLayout();
255     }
256

```

6.2.3.11 private void LIDAR_Controller.MainWindow.loadBtn_Click (object sender, RoutedEventArgs e) [private]

Event handler. Called by loadBtn for click events. Calls "DAO.loadMeasurements" to load measurements from a xml file.

1. Get a list of Measurements from a xml file
2. Clear combobox2
3. Add all measurements to combobox2
4. Reset myViewport
5. Add all measurements to myViewport as children to display them

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>sender</i>	Source of the event.
<i>e</i>	Routed event information.

Definition at line 595 of file MainWindow.xaml.cs.

```

596     {
597         //1.
598         List<Measurement> loaded = DAO.loadMeasurements();
599         if (loaded != null)
600         {
601             MeasureList = loaded;
602             //2.
603             comboBox2.Items.Clear();
604             //3.
605             foreach (Measurement x in MeasureList)
606             {
607                 comboBox2.Items.Add(x);
608             }
609             comboBox2.SelectedIndex = 0;
610             //4.
611             myViewport.Children.Clear();
612             //5.
613             myViewport.Children.Add(gridLinesXY);
614             myViewport.Children.Add(new DefaultLights());
615             foreach (Measurement x in MeasureList)
616             {
617                 x.makeGeometry3D();
618                 myViewport.Children.Add(x.getGeometry3D());
619             }
620         }
621     }

```

6.2.3.12 private void LIDAR_Controller.MainWindow.neuMessung_Click (object sender, RoutedEventArgs e) [private]

Event handler. Called by neuMessung for click events. This function adds a new [Measurement](#) to MeasureList and refreshes combobox2.

1. Add new [Measurement](#) to MeasureList
2. Clear combobox2
3. Add all Measurements to combobox2
4. Display the new [Measurement](#)

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>sender</i>	Source of the event.
<i>e</i>	Routed event information.

Definition at line 449 of file MainWindow.xaml.cs.

```

450     {
451         //1.
452         Measurement mhelper = new Measurement(new Vector3D(), 0, 0);
453         MeasureList.Add(mhelper);
454         //2.
455         comboBox2.Items.Clear();

```

```

456         //3.
457         foreach (Measurement x in MeasureList)
458         {
459             comboBox2.Items.Add(x);
460         }
461         comboBox2.SelectedIndex = 0;
462         //4.
463         myViewport.Children.Add(mhelper.getGeometry3D());
464
465     }

```

6.2.3.13 private void LIDAR_Controller.MainWindow.offsetBtn_Click (object sender, RoutedEventArgs e) [private]

Event handler. Called by offsetBtn for click events. This function gets the offset information. Then id adds this values to the selected measurement.

a. Try to parse all values

1. Set all offset values

b. If parsing fails show a MessageBox.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>sender</i>	Source of the event.
<i>e</i>	Routed event information.

Definition at line 534 of file MainWindow.xaml.cs.

```

535     {
536         double x, y, z, degX, degZ = 0;
537         //a
538         if (double.TryParse(xOffsetTxt.Text, out x) &&
539             double.TryParse(yOffsetTxt.Text, out y) &&
540             double.TryParse(zOffsetTxt.Text, out z) &&
541             double.TryParse(degOffsetXTxt.Text, out degX) &&
542             double.TryParse(degOffsetZTxt.Text, out degZ)
543         )
544         {
545             //1
546             MeasureList[selectedMeasure].linearOffset = new Vector3D(x, y, z)
547         ;
548             MeasureList[selectedMeasure].rotaryOffsetX = degX;
549             MeasureList[selectedMeasure].rotaryOffsetZ = degZ;
550             MeasureList[selectedMeasure].Refresh();
551         }
552         //b
553         else
554         {
555             MessageBox.Show("Für den Offset sind nur Zahlenwerte erlaubt.", "Fehleingabe erkannt!",
556                             MessageBoxButton.OK, MessageBoxImage.Information);
557         }
558     }

```

6.2.3.14 private void LIDAR_Controller.MainWindow.posBtn_Click (object sender, RoutedEventArgs e) [private]

Event handler. Called by posBtn for click events. This functions sends the "#3" (Set Position) command with the position infos from "txt_MPos" & "txt_SPos" to the LIDAR-Scanner.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>sender</i>	Source of the event.
<i>e</i>	Routed event information.

Definition at line 397 of file MainWindow.xaml.cs.

```
398         {
399             int m;
400             int s;
401             if (int.TryParse(txt_MPos.Text, out m) && (int.TryParse(txt_SPos.Text, out s)))
402             {
403                 ComPort.WriteLine("#3");
404                 ComPort.WriteLine("" + m);
405                 ComPort.WriteLine("" + s);
406             }
407             else
408             {
409                 MessageBox.Show("Es sind nur positive ganze Zahlen als Position erlaubt!", "Fehleingabe
erkannt!", MessageBoxButton.OK, MessageBoxImage.Information);
410             }
411         }
412     }
```

6.2.3.15 private void LIDAR_Controller.MainWindow.reverse_Click (object sender, RoutedEventArgs e) [private]

Event handler. Called by reverse for click events. This function transforms the 3D model to a open/closed shape.

Author

Alex

Date

18.01.2016

Parameters

<i>sender</i>	Source of the event.
<i>e</i>	Routed event information.

Definition at line 690 of file MainWindow.xaml.cs.

```
691         {
692             MeasureList[selectedMeasure].open = !
MeasureList[selectedMeasure].open;
693             MeasureList[selectedMeasure].Refresh();
694         }
```

6.2.3.16 private void LIDAR_Controller.MainWindow.saveBtn_Click (object sender, RoutedEventArgs e) [private]

Event handler. Called by saveBtn for click events. Calls "DAO.saveMeasurements" to save all measurements to a xml file.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>sender</i>	Source of the event.
<i>e</i>	Routed event information.

Definition at line 572 of file MainWindow.xaml.cs.

```
573         {
574             DAO.saveMeasurements(MeasureList);
575         }
```

6.2.3.17 private void LIDAR_Controller.MainWindow.sendBtn_Click (object sender, RoutedEventArgs e) [private]

Event handler. Called by sendBtn for click events. This function sends the text of "sendTxt" to the LIDAR-Scanner.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>sender</i>	Source of the event.
<i>e</i>	Routed event information.

Definition at line 636 of file MainWindow.xaml.cs.

```

637         {
638
639             ComPort.Write(sendTxt.Text);
640
641         }

```

6.2.3.18 private void LIDAR_Controller.MainWindow.verbinden_Click (object sender, RoutedEventArgs e) [private]

Event handler. Called by verbinden for click events.

a: if no problems occur then...

1. If no device is connected then ...
 - 1.1. Get baud rate from combobox1
 - 1.2. Create a new connection
 - 1.3. Register a handler for "DataReceived"-Events
 - 1.4. Open connection
 - 1.5. Clear all buffers
 - 1.6. Enable all GUI-Elements
2. else
 - 2.1. Check if the connection is still open
 - 2.2. Clear all buffers
 - 2.3. Close the connection
 - 2.4. Delete the connection
 - 2.5. Disable some GUI-Elements

b: If something goes wrong, call the exception handler.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>sender</i>	Source of the event.
<i>e</i>	Routed event information.

Definition at line 291 of file MainWindow.xaml.cs.

```

292         {
293             //a
294             try
295             {
296                 //1

```

```

297         if (ComPort == null)
298         {
299             //1.1
300             int baud;
301             Int32.TryParse(comboBox1.Text, out baud);
302             //1.2
303             ComPort = new SerialPort(comboBox.Text, baud);
304             ComPort.DtrEnable = true;
305             //1.3
306             ComPort.DataReceived += ComPortReceiveHandler;
307             //1.4
308             ComPort.Open();
309             //1.5
310             ComPort.DiscardInBuffer();
311             ComPort.DiscardOutBuffer();
312
313             textBox1.Clear();
314
315             button.Content = "Trennen";
316             label.Content = "Verbunden";
317             //1.6
318             enableVisuals(true);
319
320         }
321         //2
322         else
323         {
324             //2.1
325             if (ComPort.IsOpen)
326             {
327                 //2.2
328                 ComPort.DiscardInBuffer();
329                 ComPort.DiscardOutBuffer();
330                 //2.3
331                 ComPort.Close();
332                 //2.4
333                 ComPort = null;
334                 button.Content = "Verbinden";
335                 label.Content = "Nicht verbunden!";
336                 //2.5
337                 enableVisuals(false);
338             }
339         }
340     }
341     //b
342     catch (Exception ex)
343     {
344         ExeptionHandler(ex);
345     }
346 }

```

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

- [LIDAR_VIS_TEST/LIDAR_WPF_TEST/MainWindow.xaml.cs](#)

6.3 LIDAR_Controller.Measurement Class Reference

(Serializable) a measurement.

Public Member Functions

- [Measurement](#) ()
Default constructor.
- [Measurement](#) (Vector3D [linearOffset](#), double [rotaryOffsetX](#), double [rotaryOffsetZ](#))
Constructor. This constructor sets the initial offset values and generates a random color.
- double [getDistanceData](#) (int mpos, int spos)
Gets distance data.
- void [setDistanceData](#) (int mpos, int spos, int data)

- *Sets distance data.*
- void `makeGeometry3D` ()
Generates the 3D structure.
- ModelVisual3D `getGeometry3D` ()
Gets geometry data.
- void `setGeometryPoint3D` (int i, int k)
Relocates a specific point.
- void `Refresh` ()
Refreshes this object.
- bool `Equals` (Measurement m)
Tests if this `Measurement` is considered equal to another.

Public Attributes

- double[][] `distanceData` = new double[`maxMPos`+1][]
Information describing the distance. This array gets filled by the LIDAR-Scanner.
- Point3D `origin` = new Point3D(0, 0, 0)
The origin of the measurement.
- bool `open` = true
Generate a open or closed 3D model.
- Color `color`
The measurement color.

Static Public Attributes

- static int `maxMPos` = 200
The maximum motor position.
- static int `maxSPos` = 90
The maximum servo position.

Properties

- static int `id` [get, set]
Gets or sets the global measurement identifier.
- int `mld` [get, set]
Gets or sets the local measurement identifier.
- Vector3D `linearOffset` [get, set]
Gets or sets the linear offset.
- double `rotaryOffsetX` [get, set]
Gets or sets the rotary offset around X.
- double `rotaryOffsetZ` [get, set]
Gets or sets the rotary offset around Z.

Private Member Functions

- Vector3D `Turn3DVektorXZ` (Vector3D v, double mdegree, double sdegree)
Turns a given X vector to the specified position.

Private Attributes

- MeshGeometry3D `meshMain3D` = new MeshGeometry3D()
The mesh that contains all 3D-Data.
- ModelVisual3D `geometry3D` = new ModelVisual3D()
The geometry data.
- GeometryModel3D `geometryModel3D` = new GeometryModel3D()
The geometry model.

6.3.1 Detailed Description

(Serializable) a measurement.

Author

Alexander Miller (7089316)

Date

22.12.2015

Definition at line 29 of file Measurement.cs.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 public LIDAR_Controller.Measurement.Measurement ()

Default constructor.

Author

Alexander Miller (7089316)

Date

22.12.2015

Definition at line 154 of file Measurement.cs.

```
154 { }
```

6.3.2.2 public LIDAR_Controller.Measurement.Measurement (Vector3D *linearOffset*, double *rotaryOffsetX*, double *rotaryOffsetZ*)

Constructor. This constructor sets the initial offset values and generates a random color.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>linearOffset</i>	The linear offset.
<i>rotaryOffsetX</i>	The rotary offset around x.
<i>rotaryOffsetZ</i>	The rotary offset around z.

Definition at line 170 of file Measurement.cs.

```

171     {
172         this.linearOffset = linearOffset;
173         origin.X = linearOffset.X;
174         origin.Y = linearOffset.Y;
175         origin.Z = linearOffset.Z;
176
177         this.rotaryOffsetX = rotaryOffsetX;
178         this.rotaryOffsetZ = rotaryOffsetZ;
179         Random r = new Random();
180         this.color = Color.FromArgb(150, (Byte)r.Next(0, 256), (Byte)r.Next(0, 256), (Byte)r.Next(
0, 256));
181         mId = id++;
182
183         for (int i = 0; i < maxMPos+1; i++)
184         {
185             distanceData[i] = new double[maxSPos+1];
186             for (int k = 0; k <= maxSPos; k++)
187                 { setDistanceData(i, k, 4000); }
188         }
189         makeGeometry3D();
190     }

```

6.3.3 Member Function Documentation

6.3.3.1 public bool LIDAR_Controller.Measurement.Equals (Measurement *m*)

Tests if this [Measurement](#) is considered equal to another.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>m</i>	The measurement to compare to this object.
----------	--

Returns

true if the objects are considered equal, false if they are not.

Definition at line 417 of file Measurement.cs.

```

418     {
419         if (m == null) { return false; }
420         if (m.mId == this.mId) { return true; }
421         return false;
422     }

```

6.3.3.2 public double LIDAR_Controller.Measurement.getDistanceData (int *mpos*, int *spos*)

Gets distance data.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>mpos</i>	The motor position.
<i>spos</i>	The servo position.

Returns

The distance data.

Definition at line 206 of file Measurement.cs.

```
207         {  
208             return distanceData[mpos][spos];  
209         }
```

6.3.3.3 public ModelVisual3D LIDAR_Controller.Measurement.getGeometry3D ()

Gets geometry data.

Author

Alexander Miller (7089316)

Date

22.12.2015

Returns

The geometry data.

Definition at line 311 of file Measurement.cs.

```
312         {  
313             return geometry3D;  
314         }
```

6.3.3.4 public void LIDAR_Controller.Measurement.makeGeometry3D ()

Generates the 3D structure.

1. Add origin as first point
2. Add all points that represent the distance
 - 2.1. Turn the Vector of the actual point (distance value = x value) around the z-axis, x-axis and again around the z-axis.
 - 2.2. Add the rotated point to the model
3. Generate triangles to build the bottom/back layer
4. Generate the area between bottom and back layer

Author

Alexander Miller (7089316)

Date

22.12.2015

Definition at line 244 of file Measurement.cs.

```

245     {
246         //1.
247         meshMain3D = new MeshGeometry3D();
248         meshMain3D.Positions.Add(origin);
249         //2.
250         for (int k = 0; k <= maxSPos; k++)
251         {
252             for (int i = 0; i < maxMPos; i++)
253             {
254                 //2.1.
255                 Vector3D v = Turn3DVektorXZ(new Vector3D(
distanceData[i][k], 0, 0), i * (360.0 / maxMPos), k);
256                 //2.2.
257                 meshMain3D.Positions.Add(new Point3D(v.X +
linearOffset.X, v.Y + linearOffset.Y, v.Z + linearOffset.Z));
258             }
259         }
260
261         if (!open)
262         {
263             //3.
264             for (int i = 0; i <= maxSPos; i += maxSPos)
265             {
266                 for (int k = 1; k <= maxMPos / 2; k++)
267                 {
268                     meshMain3D.TriangleIndices.Add(0);
269                     meshMain3D.TriangleIndices.Add(k + (i * maxMPos));
270                     meshMain3D.TriangleIndices.Add(k + 1 + (i * maxMPos));
271                 }
272             }
273         }
274
275         //4.
276         for (int i = 0; i < maxSPos; i++)
277         {
278             for (int k = 1; k <= maxMPos / 2; k++)
279             {
280                 //left triangles
281                 meshMain3D.TriangleIndices.Add(k + (i * maxMPos));
282                 meshMain3D.TriangleIndices.Add(k + (i * maxMPos) + 1);
283                 meshMain3D.TriangleIndices.Add(k + ((i + 1) * maxMPos));
284                 //right triangles

```



```

287             meshMain3D.TriangleIndices.Add(k + ((i) * maxMPos) + 1);
288             meshMain3D.TriangleIndices.Add(k + ((i + 1) * maxMPos) + 1);
289             meshMain3D.TriangleIndices.Add(k + ((i + 1) * maxMPos));
290         }
291     }
292
293     geometryModel3D.Geometry = meshMain3D;
294     DiffuseMaterial matDiffuseMain = new DiffuseMaterial(new SolidColorBrush(
color));
295     geometryModel3D.Material = matDiffuseMain;
296     geometryModel3D.BackMaterial = matDiffuseMain;
297     geometry3D.Content = geometryModel3D;
298 }

```

6.3.3.5 public void LIDAR_Controller.Measurement.Refresh ()

Refreshes this object.

Author

Alexander Miller (7089316)

Date

22.12.2015

Definition at line 388 of file Measurement.cs.

```

389     {
390         makeGeometry3D();
391         origin.X = linearOffset.X;
392         origin.Y = linearOffset.Y;
393         origin.Z = linearOffset.Z;
394         for (int i = 0; i < maxMPos; i++)
395         {
396             for (int k = 0; k < maxSPos; k++)
397             {
398                 setGeometryPoint3D(i, k);
399             }
400         }
401     }
402 }

```

6.3.3.6 public void LIDAR_Controller.Measurement.setDistanceData (int mpos, int spos, int data)

Sets distance data.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>mpos</i>	The motor position.
<i>spos</i>	The servo position.
<i>data</i>	The data.

Definition at line 224 of file Measurement.cs.

```

225         {
226             distanceData[mpos][spos] = data;
227         }

```

6.3.3.7 public void LIDAR_Controller.Measurement.setGeometryPoint3D (int *i*, int *k*)

Relocates a specific point.

1. Get all points
2. Reset origin
3. Turn the Vector of the actual point (distance value = x value) around the z-axis, x-axis and again around the z-axis.
4. Set the point

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>i</i>	Zero-based index of the point (and position of the motor).
<i>k</i>	Position of the servo.

Definition at line 332 of file Measurement.cs.

```

333         {
334             //1.
335             Point3DCollection points = meshMain3D.Positions;
336
337             //2.
338             points[0] = origin;
339
340             //3.
341             Vector3D v = Turn3DVektorXZ(new Vector3D(distanceData[i][k], 0, 0), i
            * (360.0 / maxMPos), k);
342
343             //4.
344             points[i + 1 + k * maxMPos] = new Point3D(v.X + linearOffset.X, v.Y +
linearOffset.Y, v.Z + linearOffset.Z);
345
346         }

```

6.3.3.8 private Vector3D LIDAR_Controller.Measurement.Turn3DVektorXZ (Vector3D v, double mdegree, double sdegree) [private]

Turns a given X vector to the specified position.

1. Transform degrees into rad
2. Turn the vector based on the values around zxz-axis.

Author

Alexander Miller (7089316)

Date

22.12.2015

Parameters

<i>v</i>	The Vector3D to process.
<i>mdegree</i>	The motor position in degrees.
<i>sdegree</i>	The servo position in degrees.

Returns

A Vector3D.

Definition at line 365 of file Measurement.cs.

```

366     {
367         //1.
368         double mdeg = (mdegree) * (Math.PI / 180);
369         double sdeg = (sdegree + rotaryOffsetX) * (Math.PI / 180);
370         double mdeg_offset = (rotaryOffsetZ) * (Math.PI / 180);
371         //2.
372         Vector3D vn = new Vector3D();
373         vn.X = v.X * (Math.Cos(mdeg) * Math.Cos(mdeg_offset) - Math.Sin(mdeg) * Math.Cos(sdeg) * Math.
Sin(mdeg_offset)) + v.Y * (-1 * Math.Sin(mdeg) * Math.Cos(mdeg_offset) - Math.Cos(mdeg) * Math.Cos(sdeg) *
Math.Sin(mdeg_offset)) + v.Z * (Math.Sin(sdeg) * Math.Sin(mdeg_offset));
374         vn.Y = v.X * (Math.Cos(mdeg) * Math.Sin(mdeg_offset) + Math.Sin(mdeg) * Math.Cos(sdeg) * Math.
Cos(mdeg_offset)) + v.Y * (-1 * Math.Sin(mdeg) * Math.Sin(mdeg_offset) + Math.Cos(mdeg) * Math.Cos(sdeg) *
Math.Cos(mdeg_offset)) + v.Z * (Math.Sin(sdeg) * Math.Cos(mdeg_offset));
375         vn.Z = v.X * (Math.Sin(mdeg) * Math.Sin(sdeg)) + v.Y * (Math.Cos(mdeg) * Math.Sin(sdeg)) + v.Z
* (Math.Cos(sdeg));
376         return vn;
377     }

```

6.3.4 Property Documentation

6.3.4.1 public static int LIDAR_Controller.Measurement.id [static], [get], [set]

Gets or sets the global measurement identifier.

Returns

The identifier.

Definition at line 53 of file Measurement.cs.

6.3.4.2 `public Vector3D LIDAR_Controller.Measurement.linearOffset` `[get]`, `[set]`

Gets or sets the linear offset.

Returns

The linear offset in cm.

Definition at line 88 of file Measurement.cs.

6.3.4.3 `public int LIDAR_Controller.Measurement.mld` `[get]`, `[set]`

Gets or sets the local measurement identifier.

Returns

The measurement identifier.

Definition at line 63 of file Measurement.cs.

6.3.4.4 `public double LIDAR_Controller.Measurement.rotaryOffsetX` `[get]`, `[set]`

Gets or sets the rotary offset around X.

Returns

The rotary offset in degrees.

Definition at line 98 of file Measurement.cs.

6.3.4.5 `public double LIDAR_Controller.Measurement.rotaryOffsetZ` `[get]`, `[set]`

Gets or sets the rotary offset around Z.

Returns

The rotary offset in degrees.

Definition at line 108 of file Measurement.cs.

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

- LIDAR_VIS_TEST/LIDAR_WPF_TEST/[Measurement.cs](#)

Chapter 7

File Documentation

7.1 LIDAR_VIS_TEST/LIDAR_WPF_TEST/DAO.cs File Reference

Implements the dao class.

Classes

- class [LIDAR_Controller.DAO](#)

A "data access object" class. This is used to separate the file operations from the GUI logic.

Namespaces

7.1.1 Detailed Description

Implements the dao class.

7.2 LIDAR_VIS_TEST/LIDAR_WPF_TEST/MainWindow.xaml.cs File Reference

Implements the main window.xaml class. This file contains all GUI specific code (Event handler for GUI-elements).

Classes

- class [LIDAR_Controller.MainWindow](#)

The application's main form. This class contains all necessary event and exception handlers.

Namespaces

7.2.1 Detailed Description

Implements the main window.xaml class. This file contains all GUI specific code (Event handler for GUI-elements).

7.3 LIDAR_VIS_TEST/LIDAR_WPF_TEST/Measurement.cs File Reference

Implements the measurement class.

Classes

- class [LIDAR_Controller.Measurement](#)
(*Serializable*) a measurement.

Namespaces

7.3.1 Detailed Description

Implements the measurement class.

Index

- button2_Click
 - LIDAR_Controller::MainWindow, 16
- button3_Click
 - LIDAR_Controller::MainWindow, 16
- button6_Click
 - LIDAR_Controller::MainWindow, 17
- ComPortReceiveHandler
 - LIDAR_Controller::MainWindow, 18
- comboBox2_SelectionChanged
 - LIDAR_Controller::MainWindow, 17
- enableVisuals
 - LIDAR_Controller::MainWindow, 19
- entfMessung_Click
 - LIDAR_Controller::MainWindow, 20
- Equals
 - LIDAR_Controller::Measurement, 32
- ExceptionHandler
 - LIDAR_Controller::MainWindow, 21
- getDistanceData
 - LIDAR_Controller::Measurement, 32
- getGeometry3D
 - LIDAR_Controller::Measurement, 33
- id
 - LIDAR_Controller::Measurement, 37
- Init
 - LIDAR_Controller::MainWindow, 21
- Init3D
 - LIDAR_Controller::MainWindow, 22
- LIDAR_Controller, 9
- LIDAR_Controller.DAO, 11
- LIDAR_Controller.MainWindow, 14
- LIDAR_Controller.Measurement, 29
- LIDAR_Controller::DAO
 - loadMeasurements, 12
 - saveMeasurements, 12
- LIDAR_Controller::MainWindow
 - button2_Click, 16
 - button3_Click, 16
 - button6_Click, 17
 - ComPortReceiveHandler, 18
 - comboBox2_SelectionChanged, 17
 - enableVisuals, 19
 - entfMessung_Click, 20
 - ExceptionHandler, 21
 - Init, 21
 - Init3D, 22
 - loadBtn_Click, 23
 - MainWindow, 16
 - neuMessung_Click, 24
 - offsetBtn_Click, 25
 - posBtn_Click, 25
 - reverse_Click, 26
 - saveBtn_Click, 27
 - sendBtn_Click, 27
 - verbinden_Click, 28
- LIDAR_Controller::Measurement
 - Equals, 32
 - getDistanceData, 32
 - getGeometry3D, 33
 - id, 37
 - linearOffset, 37
 - mId, 38
 - makeGeometry3D, 33
 - Measurement, 31
 - Refresh, 35
 - rotaryOffsetX, 38
 - rotaryOffsetZ, 38
 - setDistanceData, 35
 - setGeometryPoint3D, 36
 - Turn3DVektorXZ, 36
- LIDAR_VIS_TEST/LIDAR_WPF_TEST/DAO.cs, 39
- LIDAR_VIS_TEST/LIDAR_WPF_TEST/MainWindow.↔
 - xaml.cs, 39
- LIDAR_VIS_TEST/LIDAR_WPF_TEST/Measurement.↔
 - cs, 40
- linearOffset
 - LIDAR_Controller::Measurement, 37
- loadBtn_Click
 - LIDAR_Controller::MainWindow, 23
- loadMeasurements
 - LIDAR_Controller::DAO, 12
- mId
 - LIDAR_Controller::Measurement, 38
- MainWindow
 - LIDAR_Controller::MainWindow, 16
- makeGeometry3D
 - LIDAR_Controller::Measurement, 33
- Measurement
 - LIDAR_Controller::Measurement, 31
- neuMessung_Click
 - LIDAR_Controller::MainWindow, 24
- offsetBtn_Click
 - LIDAR_Controller::MainWindow, 25

posBtn_Click
 LIDAR_Controller::MainWindow, [25](#)

Refresh
 LIDAR_Controller::Measurement, [35](#)

reverse_Click
 LIDAR_Controller::MainWindow, [26](#)

rotaryOffsetX
 LIDAR_Controller::Measurement, [38](#)

rotaryOffsetZ
 LIDAR_Controller::Measurement, [38](#)

saveBtn_Click
 LIDAR_Controller::MainWindow, [27](#)

saveMeasurements
 LIDAR_Controller::DAO, [12](#)

sendBtn_Click
 LIDAR_Controller::MainWindow, [27](#)

setDistanceData
 LIDAR_Controller::Measurement, [35](#)

setGeometryPoint3D
 LIDAR_Controller::Measurement, [36](#)

Turn3DVektorXZ
 LIDAR_Controller::Measurement, [36](#)

verbinden_Click
 LIDAR_Controller::MainWindow, [28](#)