LIDAR Controller

v1.0

Generated by Doxygen 1.8.11

Contents

Chapter 1

Namespace Index

1.1 Packages

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

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

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 ssk 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.

```
        {
            if (MessageBox.Show("Möchten Sie Messergebnisse aus einer Datei laden?\nDabei gehen nicht
 gespeicherte Messungen verloren!", "Warnung!", MessageBoxButton.OKCancel, MessageBoxImage.Warning) ==
MessageBoxResult.OK)
            {
                List<Measurement> list;
                //a.
                try
                {
                    //1.
                    OpenFileDialog ofdialog = new OpenFileDialog();
                    ofdialog.DefaultExt = ".lmd"; //LIDAR Measurement Data
                    ofdialog.Filter = "LIDAR Measurement Data|*.lmd";
                    Nullable<bool> r = ofdialog.ShowDialog();
                    if (r == true)
                    {
                        //2.
                        FileStream fs = new FileStream(ofdialog.FileName, FileMode.Open);
                        //3.
                        Type t = typeof(List<Measurement>);
                        XmlSerializer serializer = new XmlSerializer(t);
                        //4.
                        list = (List<Measurement>)serializer.Deserialize(fs);
                        //5.
                        Measurement.id = list[list.Count-1].mId+1;
                        //6.
                        fs.Close();
                        //7.
                        MessageBox.Show("Laden erfolgreich.", "Info", MessageBoxButton.OK, MessageBoxImage.
Information);
                        return list;
                    }

                }
                //b.
                catch (Exception E)
                {
                    MessageBox.Show(E.Message, "Fehler!", MessageBoxButton.OK, MessageBoxImage.Error);
                }
            }
            return null;
            }
```

6.1.2.2 public static void LIDAR_Controller.DAO.saveMeasurements (List< Measurement > *l*) [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

l	The List<Measurement> to process.

Definition at line 120 of file DAO.cs.

```
        {
            //a.
            try
            {
                //1.
                SaveFileDialog sfdialog = new SaveFileDialog();
                sfdialog.DefaultExt = ".lmd"; //LIDAR Measurment Data
                sfdialog.Filter = "LIDAR Measurment Data|*.lmd";
                Nullable<bool> r = sfdialog.ShowDialog();
                if (r == true)
                {
                    XmlWriterSettings xmlWriterSettings = new XmlWriterSettings();
                    xmlWriterSettings.Indent = true;
                    //2.
                    FileStream fs = new FileStream(sfdialog.FileName, FileMode.Create);
                    //3.
                    Type t = typeof(List<Measurement>);
                    XmlSerializer serializer = new XmlSerializer(t);
                    //4.
                    XmlWriter xmlWriter = XmlWriter.Create(fs, xmlWriterSettings);
                    serializer.Serialize(xmlWriter, l);
                    //5.
                    fs.Close();
                    //6.
                    MessageBox.Show("Speichern erfolgreich.", "Info", MessageBoxButton.OK, MessageBoxImage.
    Information);
                }

            }
            //b.
            catch (Exception E)
            {
                MessageBox.Show(E.Message, "Fehler!", MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }
```

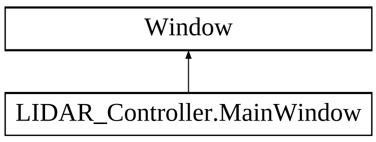
The documentation for this class was generated from the following file:

- 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 id 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.

```
        {
            InitializeComponent();
            Init();
            Init3D();
        }
```

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

sender	Source of the event.
e	Routed event information.

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

```
        {
            ComPort.WriteLine("#1");
        }
```

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

sender	Source of the event.
e	Routed event information.

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

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

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

sender	Source of the event.
e	Routed event information.

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

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

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

Sender	Source of the event.
e	Selection changed event information.

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

```
        {
            selectedMeasure = comboBox2.SelectedIndex;
            if (selectedMeasure >= 0)
            {
                xOffsetTxt.Text = "" + MeasureList[selectedMeasure].linearOffset.
    X;
                yOffsetTxt.Text = "" + MeasureList[selectedMeasure].linearOffset.
    Y;
                zOffsetTxt.Text = "" + MeasureList[selectedMeasure].linearOffset.
    Z;
                degOffsetXTxt.Text = "" + MeasureList[selectedMeasure].
    rotaryOffsetX;
                degOffsetZTxt.Text = "" + MeasureList[selectedMeasure].
    rotaryOffsetZ;
            }

        }
```

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

Sender	Source of the event.
e	Serial data received event information.

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

```
        {
            //a
            try
            {
                //1
                if (!ComPort.IsOpen) return;
                string ReceivedText = "";
                //2
                ReceivedText = ComPort.ReadLine();
                //3
                Dispatcher.BeginInvoke(new Action(() =>
                {
                    this.textBox1.AppendText(ReceivedText);
                    this.textBox1.ScrollToEnd();
                }));
                //4
                string[] numbers = Regex.Split(ReceivedText, @"\D+");
                //5
                if (numbers.Length == 5)
                {
                    int mpos = 0;
                    int spos = 0;
                    int value = 0;
                    if (int.TryParse(numbers[1], out mpos) && int.TryParse(numbers[2], out spos) && int.TryParse(numbers[3],out value))
                        //6
                        MeasureList[selectedMeasure].setDistanceData(mpos+1, spos, value);
                    Dispatcher.BeginInvoke(new Action(() =>
                    {
                        MeasureList[selectedMeasure].setGeometryPoint3D(mpos+1 , spos);

                    }));



                }
             }
             //b
             catch (Exception ex) { ExeptionHandler(ex); }
         }
```

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

b	true enables visuals and false disables them.

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

```
        {
            //neuMessung.IsEnabled = b;
            //entfMessung.IsEnabled = b;
            //offsetBtn.IsEnabled = b;
            btn_einzelmessung.IsEnabled = b;
            btn_kalibrieren.IsEnabled = b;
            posBtn.IsEnabled = b;
            btn_radar.IsEnabled = b;
            txt_MPos.IsEnabled = b;
            txt_SPos.IsEnabled = b;
            //sendBtn.IsEnabled = b;
            //comboBox2.IsEnabled = b;
            //sendTxt.IsEnabled = b;
            //txt_punktmessung.IsEnabled = b;
            //xOffsetTxt.IsEnabled = b;
            //yOffsetTxt.IsEnabled = b;
            //zOffsetTxt.IsEnabled = b;
            //degOffsetTxt.IsEnabled = b;
            //textBox1.IsEnabled = b;
        }
```

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

sender	Source of the event.
e	Routed event information.

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

```
        {
            //1
            if (MeasureList.Count > 1)
            {
                //2
                MeasureList.RemoveAt(selectedMeasure);
                //3
                comboBox2.Items.Clear();
                //4
                foreach (Measurement x in MeasureList)
                {
                    comboBox2.Items.Add(x);
                }
                comboBox2.SelectedIndex = 0;
                //5
                myViewport.Children.Clear();
                //6
                myViewport.Children.Add(gridLinesXY);
                myViewport.Children.Add(new DefaultLights());
                foreach (Measurement x in MeasureList)
                {
                    myViewport.Children.Add(x.getGeometry3D());
                }
            }


        }
```

6.2.3.8 private void LIDAR_Controller.MainWindow.ExeptionHandler (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.

```
        {
            switch (ex.GetType().Name)
            {
                case "IOException":
                    MessageBox.Show("Name: " + ex.GetType().Name + "\r\nBeschreibung:\r\n" + ex.Message, "
    Fehler!", MessageBoxButton.OK, MessageBoxImage.Error);
                    break;
                case "ArgumentException":
                    MessageBox.Show("Bitte Eingaben überprüfen!", "Fehleingabe erkannt!", MessageBoxButton.
    OK, MessageBoxImage.Information);
                    break;
                default:
                    MessageBox.Show("Name: " + ex.GetType().Name + "\r\nBeschreibung:\r\n" + ex.Message, "
    Unbekannter Fehler!", MessageBoxButton.OK, MessageBoxImage.Error);
                    break;
            }

        }
```

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.

```
        {
            //add all COM-Ports to combobox
            foreach (string ports in SerialPort.GetPortNames())
            {
                comboBox.Items.Add(ports);
            }
            //initial measurement
            MeasureList.Add(new Measurement(new Vector3D(0, 0, 0.5), 0, 0));

            //add measurement to combobox2 and register a new event handler for "SelectionChangedEvent"
            comboBox2.Items.Add(MeasureList[selectedMeasure]);
            comboBox2.DisplayMemberPath = "mId";
            comboBox2.SelectionChanged += new SelectionChangedEventHandler(
      comboBox2_SelectionChanged);
            comboBox2.SelectedIndex = 0;
            //Enable/Disable all GUI-Elements that need a connected COM-Port
            enableVisuals(false);

        }
```

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.

```
        {
            //1
            gridLinesXY.MinorDistance = 10; //Distance in cm for the minor lines
            gridLinesXY.MajorDistance = 100; //Distance in cm for the major lines
            gridLinesXY.Thickness = 1.5; //Line thickness
            gridLinesXY.Length = 10000; //Maximum grid size
            gridLinesXY.Width = 10000; //Maximum grid size

            //2
            myViewport.ShowFrameRate = true; //Show framerate
            myViewport.ShowCoordinateSystem = true; //Show small Coordinate system
            myViewport.ShowFieldOfView = true; //Show Field of View

            //3
            myViewport.Children.Add(gridLinesXY);
            myViewport.Children.Add(new DefaultLights());
            myViewport.Children.Add(MeasureList[
      selectedMeasure].getGeometry3D());

            //4
            grid.Children.Add(myViewport);
            grid.UpdateLayout();

        }
```

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

sender	Source of the event.
e	Routed event information.

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

```
        {
            //1.
            List<Measurement> loaded = DAO.loadMeasurements();
            if (loaded != null)
            {
                MeasureList = loaded;
                //2.
                comboBox2.Items.Clear();
                //3.
                foreach (Measurement x in MeasureList)
                {
                    comboBox2.Items.Add(x);
                }
                comboBox2.SelectedIndex = 0;
                //4.
                myViewport.Children.Clear();
                //5.
                myViewport.Children.Add(gridLinesXY);
                myViewport.Children.Add(new DefaultLights());
                foreach (Measurement x in MeasureList)
                {
                    x.makeGeometry3D();
                    myViewport.Children.Add(x.getGeometry3D());
                }
            }
        }
```

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

sender	Source of the event.
e	Routed event information.

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

```
        {
            //1.
            Measurement mhelper = new Measurement(new Vector3D(), 0, 0);
            MeasureList.Add(mhelper);
            //2.
            comboBox2.Items.Clear();
            //3.
            foreach (Measurement x in MeasureList)
            {
                comboBox2.Items.Add(x);
            }
            comboBox2.SelectedIndex = 0;
            //4.
            myViewport.Children.Add(mhelper.getGeometry3D());

        }
```

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

sender	Source of the event.
e	Routed event information.

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

```
        {
            double x, y, z, degX ,degZ = 0;
            //a
            if (double.TryParse(xOffsetTxt.Text, out x) &&
                double.TryParse(yOffsetTxt.Text, out y) &&
                double.TryParse(zOffsetTxt.Text, out z) &&
                double.TryParse(degOffsetXTxt.Text, out degX) &&
                double.TryParse(degOffsetZTxt.Text, out degZ)
                )
            {
                //1
                MeasureList[selectedMeasure].linearOffset = new Vector3D(x, y, z)
;
                MeasureList[selectedMeasure].rotaryOffsetX = degX;
                MeasureList[selectedMeasure].rotaryOffsetZ = degZ;
                MeasureList[selectedMeasure].Refresh();

            }
            //b
            else
            {
                MessageBox.Show("Für den Offset sind nur Zahlenwerte erlaubt.", "Fehleingabe erkannt!",
MessageBoxButton.OK, MessageBoxImage.Information);
            }
        }
```

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

sender	Source of the event.
e	Routed event information.

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

```
        {
            int m;
            int s;
            if (int.TryParse(txt_MPos.Text, out m) && (int.TryParse(txt_SPos.Text, out s)))
            {
                ComPort.WriteLine("#3");
                ComPort.WriteLine("" + m);
                ComPort.WriteLine("" + s);
            }
            else
            {
                MessageBox.Show("Es sind nur positive ganze Zahlen als Position erlaubt!", "Fehleingabe erkannt!", MessageBoxButton.OK, MessageBoxImage.Information);
            }

        }
```

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

sender	Source of the event.
e	Routed event information.

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

```
        {
            MeasureList[selectedMeasure].open = !
    MeasureList[selectedMeasure].open;
            MeasureList[selectedMeasure].Refresh();
        }
```

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

sender	Source of the event.
e	Routed event information.

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

```
        {
            DAO.saveMeasurements(MeasureList);
        }
```

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

sender	Source of the event.
e	Routed event information.

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

```
        {

            ComPort.Write(sendTxt.Text);

        }
```

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

sender	Source of the event.
e	Routed event information.

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

```
        {
            //a
            try
            {
                //1
                if (ComPort == null)
                {
                    //1.1
                    int baud;
                    Int32.TryParse(comboBox1.Text, out baud);
                    //1.2
                    ComPort = new SerialPort(comboBox.Text, baud);
                    ComPort.DtrEnable = true;
                    //1.3
                    ComPort.DataReceived += ComPortReceiveHandler;
                    //1.4
                    ComPort.Open();
                    //1.5
                    ComPort.DiscardInBuffer();
                    ComPort.DiscardOutBuffer();

                    textBox1.Clear();

                    button.Content = "Trennen";
                    label.Content = "Verbunden";
                    //1.6
                    enableVisuals(true);

                }
                //2
                else
                {
                    //2.1
                    if (ComPort.IsOpen)
                    {
                        //2.2
                        ComPort.DiscardInBuffer();
                        ComPort.DiscardOutBuffer();
                        //2.3
                        ComPort.Close();
                        //2.4
                        ComPort = null;
                        button.Content = "Verbinden";
                        label.Content = "Nicht verbunden!";
                        //2.5
                        enableVisuals(false);
                    }
                }
            }
            //b
            catch (Exception ex)
            {
                ExeptionHandler(ex);
            }
        }
```

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 mId `[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

linearOffset	The linear offset.
rotaryOffsetX	The rotary offset around x.
rotaryOffsetZ	The rotary offset around z.

Definition at line 170 of file Measurement.cs.

```
        {
            this.linearOffset = linearOffset;
            origin.X = linearOffset.X;
            origin.Y = linearOffset.Y;
            origin.Z = linearOffset.Z;

            this.rotaryOffsetX = rotaryOffsetX;
            this.rotaryOffsetZ = rotaryOffsetZ;
            Random r = new Random();
            this.color = Color.FromArgb(150, (Byte)r.Next(0, 256), (Byte)r.Next(0, 256), (Byte)r.Next(
    0, 256));
            mId = id++;

            for (int i = 0; i < maxMPos+1; i++)
            {
                distanceData[i] = new double[maxSPos+1];
                for (int k = 0; k <= maxSPos; k++)
                { setDistanceData(i, k, 4000); }
            }
            makeGeometry3D();
        }
```

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

m	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.

```
        {
            if (m == null) { return false; }
            if (m.mId == this.mId) { return true; }
            return false;
        }
```

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

mpos	The motor position.
spos	The servo position.

Returns

The distance data.

Definition at line 206 of file Measurement.cs.

```
        {
            return distanceData[mpos][spos];
        }
```

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.

```
        {
            return geometry3D;
        }
```

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.

```
        {
            //1.
            meshMain3D = new MeshGeometry3D();
            meshMain3D.Positions.Add(origin);
            //2.
            for (int k = 0; k <= maxSPos; k++)
            {
                for (int i = 0; i < maxMPos; i++)
                {
                    //2.1.
                    Vector3D v = Turn3DVektorXZ(new Vector3D(distanceData[i][k], 0, 0), i * (360.0 / maxMPos), k);
                    //2.2.
                    meshMain3D.Positions.Add(new Point3D(v.X + linearOffset.X, v.Y + linearOffset.Y, v.Z + linearOffset.Z));
                }
            }

            if (!open)
            {
                //3.
                for (int i = 0; i <= maxSPos; i += maxSPos)
                {
                    for (int k = 1; k <= maxMPos / 2; k++)
                    {
                        meshMain3D.TriangleIndices.Add(0);
                        meshMain3D.TriangleIndices.Add(k + (i * maxMPos));
                        meshMain3D.TriangleIndices.Add(k + 1 + (i * maxMPos));
                    }

                }
            }


            //4.
            for (int i = 0; i < maxSPos; i++)
            {
                for (int k = 1; k <= maxMPos / 2; k++)
                {
                    //left triangles
                    meshMain3D.TriangleIndices.Add(k + (i * maxMPos));
                    meshMain3D.TriangleIndices.Add(k + (i * maxMPos) + 1);
                    meshMain3D.TriangleIndices.Add(k + ((i + 1) * maxMPos));
                    //right triangles
                    meshMain3D.TriangleIndices.Add(k + ((i) * maxMPos) + 1);
                    meshMain3D.TriangleIndices.Add(k + ((i + 1) * maxMPos) + 1);
                    meshMain3D.TriangleIndices.Add(k + ((i + 1) * maxMPos));
                }
            }

            geometryModel3D.Geometry = meshMain3D;
            DiffuseMaterial matDiffuseMain = new DiffuseMaterial(new SolidColorBrush(
      color));
            geometryModel3D.Material = matDiffuseMain;
            geometryModel3D.BackMaterial = matDiffuseMain;
            geometry3D.Content = geometryModel3D;
        }
```

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.

```
        {
            makeGeometry3D();
            origin.X = linearOffset.X;
            origin.Y = linearOffset.Y;
            origin.Z = linearOffset.Z;
            for (int i = 0; i < maxMPos; i++)
            {
                for (int k = 0; k < maxSPos; k++)
                {
                    setGeometryPoint3D(i, k);
                }

            }
        }
```

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

mpos	The motor position.
spos	The servo position.
data	The data.

Definition at line 224 of file Measurement.cs.

```
        {
            distanceData[mpos][spos] = data;
        }
```

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

Definition at line 332 of file Measurement.cs.

```
        {
            //1.
            Point3DCollection points = meshMain3D.Positions;

            //2.
            points[0] = origin;

            //3.
            Vector3D v = Turn3DVektorXZ(new Vector3D(distanceData[i][k], 0, 0), i
     * (360.0 / maxMPos), k);

            //4.
            points[i + 1 + k * maxMPos] = new Point3D(v.X + linearOffset.X, v.Y +
    linearOffset.Y, v.Z + linearOffset.Z);

        }
```

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

v	The Vector3D to process.
mdegree	The motor position in degrees.
sdegree	The servo position in degrees.

Returns

A Vector3D.

Definition at line 365 of file Measurement.cs.

```
        {
            //1.
            double mdeg = (mdegree) * (Math.PI / 180);
            double sdeg = (sdegree + rotaryOffsetX) * (Math.PI / 180);
            double mdeg_offset = (rotaryOffsetZ) * (Math.PI / 180);
            //2.
            Vector3D vn = new Vector3D();
            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));
            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));
            vn.Z = v.X * (Math.Sin(mdeg) * Math.Sin(sdeg)) + v.Y * (Math.Cos(mdeg) * Math.Sin(sdeg)) + v.Z
    * (Math.Cos(sdeg));
            return vn;
        }
```

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.mId `[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