Student: Brian Johnston

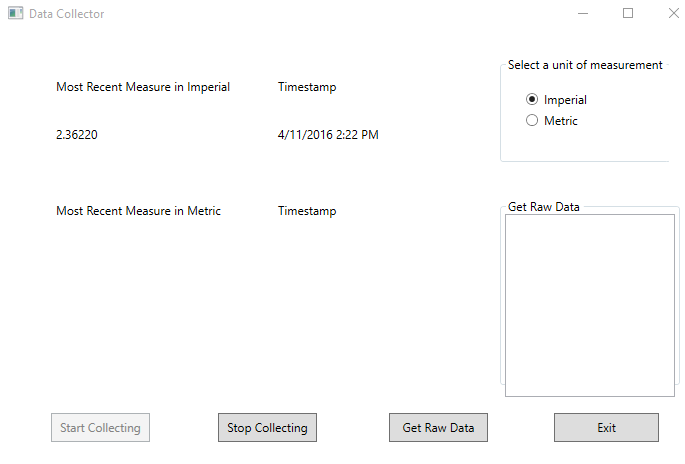
Class COP2362

Assignment: Week 6 Data collector

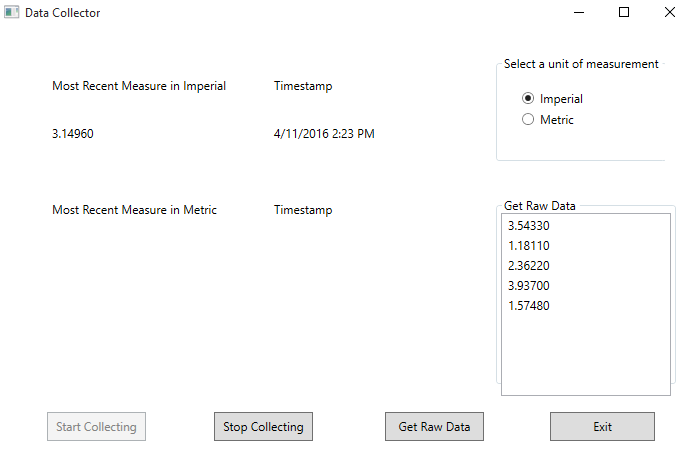
Collaboration Statement: I worked alone

Print screen:

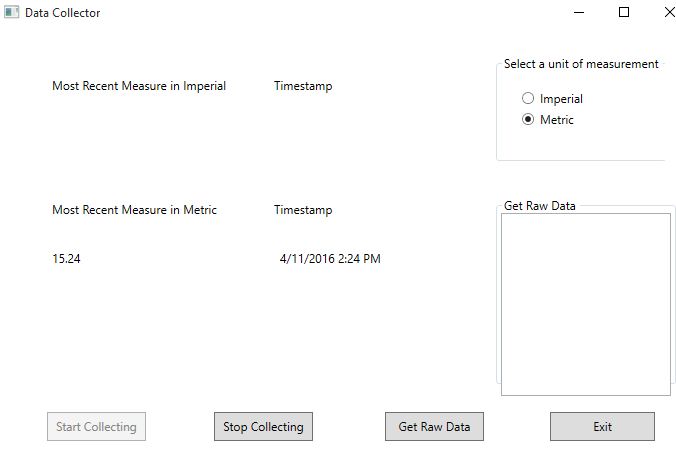
1. Capturing Data showing the active ‘Get Data’ and the most recent collection in a label on the form with a timestamp.



1. Displaying raw data in the array with a click event from a button on the main form.



Displaying data converted to the proper measurement as selected by the user.   This should be the show the opposite of the current measurement. *For our example, always assume the data coming from the ‘device’ is in inches and we want to convert to centimeters*

1. 

Code:

MainWIndow.xaml.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace DataCollector

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

//Create an instance of the MeasureLengthDevice class

MeasureLengthDevice mld = new MeasureLengthDevice();

public MainWindow()

{

InitializeComponent();

DataContext = mld;

}

//Expose the radio buttons to use on other classes

public RadioButton ImperialBtn { get { return imperialRadioBtn; } }

public RadioButton MetricBtn { get { return metricRadioBtn; } }

/// <summary>

/// When form is loaded

///

/// Certain things should happen here, rather than in the form's constructor.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void OnFormLoaded(object sender, RoutedEventArgs e)

{

// Initialize the radio buttons

if (mld.Units == UnitsEnumeration.Imperial)

{

ImperialBtn.IsChecked = true;

MetricBtn.IsChecked = false;

metricMRM.Visibility = System.Windows.Visibility.Hidden;

metricTimestamp.Visibility = System.Windows.Visibility.Hidden;

imperialMRM.Visibility = System.Windows.Visibility.Visible;

imperialTimestamp.Visibility = System.Windows.Visibility.Visible;

}

else

{

MetricBtn.IsChecked = true;

ImperialBtn.IsChecked = false;

imperialMRM.Visibility = System.Windows.Visibility.Hidden;

imperialTimestamp.Visibility = System.Windows.Visibility.Hidden;

metricMRM.Visibility = System.Windows.Visibility.Visible;

metricTimestamp.Visibility = System.Windows.Visibility.Visible;

}

}

private void startCollecting\_Click(object sender, RoutedEventArgs e)

{

startCollecting.IsEnabled = false; //Disable start button

stopCollecting.IsEnabled = true; //Enable stop button

mld.StartCollecting(); //Start Collecting

}

private void stopCollecting\_Click(object sender, RoutedEventArgs e)

{

//

startCollecting.IsEnabled = true; //Enable start button

stopCollecting.IsEnabled = false; //Disable stop button

mld.StopCollecting(); //Stop Collecting

}

private void getRawData\_Click(object sender, RoutedEventArgs e)

{

decimal[] grd = mld.GetRawData(); //Get raw data from int array

getRawDataListBox.Items.Clear(); // Clear out the message box

//Step through array and add items to Listbox

for (int i = 0; i < grd.Length; i++)

{

if (grd[i] != 0)

{

getRawDataListBox.Items.Add(grd[i].ToString());

}

}

}

//Exit button

private void exit\_Click(object sender, RoutedEventArgs e)

{

this.Close(); //Close the form

}

private void imperialRadioBtn\_Checked(object sender, RoutedEventArgs e)

{

MeasureLengthDevice data = (MeasureLengthDevice)DataContext;

metricMRM.Visibility = System.Windows.Visibility.Hidden;

metricTimestamp.Visibility = System.Windows.Visibility.Hidden;

imperialMRM.Visibility = System.Windows.Visibility.Visible;

imperialTimestamp.Visibility = System.Windows.Visibility.Visible;

}

private void metricRadioBtn\_Checked(object sender, RoutedEventArgs e)

{

MeasureLengthDevice data = (MeasureLengthDevice)DataContext;

imperialMRM.Visibility = System.Windows.Visibility.Hidden;

imperialTimestamp.Visibility = System.Windows.Visibility.Hidden;

metricMRM.Visibility = System.Windows.Visibility.Visible;

metricTimestamp.Visibility = System.Windows.Visibility.Visible;

}

/// <summary>

/// OnClosed event for MainWindow

/// </summary>

/// <param name="sender">[ignore]</param>

/// <param name="e">[ignore]</param>

private void OnClosed(object sender, EventArgs e)

{

Application.Current.Shutdown();

}

}

}

Device.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataCollector

{

public class Device

{

//Method that generates random int between 1 and 10 as a measurement of some imaginary object and returns

public int GetMeasurement()

{

Random rnd = new Random();

int measurement = rnd.Next(1, 11);

return measurement;

}

}

}

imeasuringDevices.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataCollector

{

interface IMeasuringDevice

{

/// <summary>

/// Public method declaration that represents a metric value

/// </summary>

/// <returns>Decimal of most recent capture</returns>

decimal MetricValue(int mostRecentMeasure);

/// <summary>

/// Public method declaration that represents an imperial value

/// </summary>

/// <returns>Decimal of the most recent capture</returns>

decimal ImperialValue(int mostRecentMeasure);

/// <summary>

/// Public method declaration to start device and returns nothing

/// </summary>

void StartCollecting();

/// <summary>

/// Public method declaration that stops device and returns nothing

/// </summary>

void StopCollecting();

/// <summary>

/// Public method declaration that will retrieve a copy of all the recent data the measuring device has captured

/// </summary>

/// <returns>An array of ints</returns>

decimal[] GetRawData();

}

}

MeasureDeviceLength.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Windows; // Add a reference to Windows for the hack in

using System.Threading.Tasks;

using System.Windows.Threading;

namespace DataCollector

{

public class MeasureLengthDevice : IMeasuringDevice, INotifyPropertyChanged

{

//Declare fields

DispatcherTimer timer;

private UnitsEnumeration unitsToUse;

private decimal[] dataCaptured = new decimal[] { 0, 0, 0, 0, 0 };

private int mostRecentMeasure;

public int MostRecentMeasure

{

get { return mostRecentMeasure; }

set

{

mostRecentMeasure = value;

MostRecentImperial = ImperialValue(mostRecentMeasure);

MostRecentMetric = MetricValue(mostRecentMeasure);

}

}

private Device device;

/// <summary>

/// Units property -- get only

/// </summary>

public UnitsEnumeration Units

{

get { return unitsToUse; }

}

private DateTime currentDate;

public DateTime CurrentDate

{

get { return currentDate; }

set

{

currentDate = value;

OnPropertyChanged("CurrentDate");

}

}

private decimal mostRecentImperial;

public decimal MostRecentImperial

{

get { return mostRecentImperial; }

set

{

mostRecentImperial = value;

OnPropertyChanged("MostRecentImperial");

}

}

private decimal mostRecentMetric;

public decimal MostRecentMetric

{

get { return mostRecentMetric; }

set

{

mostRecentMetric = value;

OnPropertyChanged("MostRecentMetric");

}

}

//Default Constructor to initialize fields

public MeasureLengthDevice()

{

timer = new DispatcherTimer();

timer.Tick += timer\_Tick;

unitsToUse = UnitsEnumeration.Imperial;

device = new Device();

MostRecentMeasure = device.GetMeasurement();

}

public decimal MetricValue(int mostRecentMeasure)

{

decimal mrm = Convert.ToDecimal(mostRecentMeasure);

const decimal conversion = 2.54M;

//Convert mostRecentMeasure to centimeters

mrm \*= conversion;

return mrm;

}

public decimal ImperialValue(int mostRecentMeasure)

{

decimal mrm = Convert.ToDecimal(mostRecentMeasure);

const decimal conversion = 0.39370M;

//Convert mostRecentMeasure to inches

mrm \*= conversion;

return mrm;

}

public void StartCollecting()

{

timer.Interval = new System.TimeSpan(0, 0, 15); //tick every 15 seconds

timer.Start(); //Start timer

}

//Add mostRecentMesaure to the dataCaptured int array

private void timer\_Tick(object sender, EventArgs e)

{

MainWindow main = (MainWindow) Application.Current.MainWindow; // Get the MainWindow object

MostRecentMeasure = device.GetMeasurement(); //Run device to get measurement

if (main.imperialRadioBtn.IsChecked == true) //Check user input

{

unitsToUse = UnitsEnumeration.Imperial; //Set unitsToUse

//Add data to array

for (int i = 0; i < dataCaptured.Length; i++)

{

CurrentDate = DateTime.Now;

decimal recentCapture = ImperialValue(MostRecentMeasure); //Convert

if (dataCaptured[i] == 0)

{

dataCaptured[i] = recentCapture;

break;

}

}

}

else

{

unitsToUse = UnitsEnumeration.Metric; //Set unitsToUse

//Add data to array

for (int i = 0; i < dataCaptured.Length; i++)

{

CurrentDate = DateTime.Now;

decimal recentCapture = MetricValue(MostRecentMeasure); //Convert

if (dataCaptured[i] == 0)

{

dataCaptured[i] = recentCapture;

break;

}

}

}

}

//Stop Timer

public void StopCollecting()

{

timer.Stop();

}

//Return dataCaptured int array

public decimal[] GetRawData()

{

return dataCaptured;

}

public event PropertyChangedEventHandler PropertyChanged;

private void OnPropertyChanged(string propertyName)

{

PropertyChangedEventHandler handler = PropertyChanged;

if (handler != null)

{

handler(this, new PropertyChangedEventArgs(propertyName));

}

}

}

}

UnitsEnumeration.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataCollector

{

//Public enumeration to symbolically represent the units Metric and Imperial

public enum UnitsEnumeration

{

//Values to represent Metric and Imperial units

Metric,

Imperial

}

}