**Den Ablauf des Programms.(„UniversalServer“ projekt)**

This Document is about ‚Aktionssequenz‘.(Instace of A class make instance of B class, and the instance of B class call the other class or method of the class… )

***„Ich wünsche, dass Sie sich viel Mühe geben und Ihnen gute Ergebnisse bekommen.“ –Kyuhyun Jo***

1. **MainWindow.xaml, zeile 13.**

<i:InvokeCommandAction Command="{Binding WindowLoaded}" />

**1-1.MainViewModel.cs, zeile 173.**

public ICommand WindowLoaded

{

get

{

if (\_windowLoadedCommand == null)

{

\_windowLoadedCommand = new RelayCommand(c => ExecuteWindowLoadedCommand());

}

return \_windowLoadedCommand;

}

}

**1-1-1. RelayCommand.cs, zeile 13.**

public class RelayCommand : ICommand

{

private readonly Action<object> \_execute;

private readonly Predicate<object> \_canExecute;

public RelayCommand(Action<object> execute) : this(execute, null)

{

}

public RelayCommand(Action<object> execute, Predicate<object> canExecute)

{

if (execute == null)

{

throw new ArgumentNullException("execute");

}

this.\_execute = execute;

this.\_canExecute = canExecute;

}

[DebuggerStepThrough]

public bool CanExecute(object parameter)

{

if (\_canExecute == null)

{

return true;

}

return \_canExecute(parameter);

}

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

public void Execute(object parameter)

{

\_execute(parameter);

}

}

**1-1-2. MainViewModel.cs, zeile 186.**

private void ExecuteWindowLoadedCommand()

{

//Nach den Laden des Windows, den Server starten.

//\_serv = new Server();

\_serv = new ServerMockUp(); //zum Testen kann auch ein MockUp als Datenquelle verwendet werden.

\_serv.StatusPropertyChanged += Serv\_StatusPropertyChanged;

\_serv.MessageReceived += \_serv\_MessageReceived;

\_serv.Start();

OnPropertyChanged("MessageReceived");

}

**1-1-2-1. ServerMockUp.cs, zeile 15.**

public class ServerMockUp : IServerContract

{

**public event StatusChangedEventHandler StatusPropertyChanged;**

**public event MessageReceivedEventHandler MessageReceived;**

Timer \_tmr;

**public void Start()**

{

StatusPropertyChanged("Starting Server...");

Thread.Sleep(500); //Verzögerung simulieren, wenn wir später auf echte Sockets gehen.

//Timer starten, der uns zyklisch Werte liefert.

\_tmr = new Timer(new TimerCallback(TimerProc));

\_tmr.Change(1000, 2000);

StatusPropertyChanged("Waiting for Connection...");

}

private void TimerProc(object state)

{

// The state object is the Timer object.

Timer t = (Timer)state;

//t.Dispose();

//Protokoll to simulate: Temperatur;Luftfeuchte;Luftdruck;LUX;IR

//String dataSend = String((\_temperatur + \_temp) / 2) + ";" + String(\_humidity) + ";" + String(\_press) + ";" + String(tsl.calculateLux(\_full, \_ir)) + ";" + String(\_ir);

Random rndm = new Random();

double temp = 22 + rndm.NextDouble() - rndm.NextDouble();

double hum = 50 + rndm.Next(-5, 5);

int press = 1024 + rndm.Next(-20, 20);

string ip = "192.168.1.145";

string data =

String.Format("{0:0.00}", temp) + ";" +

String.Format("{0:00.00}", hum) + ";" +

String.Format("{0:0000}", press) + ";" +

ip;

MessageReceived(data);

}

}

**1-1-2-2. MainViewModel.cs, zeile 200.**

private void Serv\_StatusPropertyChanged(string s)

{

Status = s;

}

**1-1-2-3. MainViewModel.cs, zeile 206.**

private void \_serv\_MessageReceived(string msg)

{

Status = msg;

//Message auf analysieren und auf die Eigenschaften verteilen.

//Protokoll-Format: Temperatur;Luftfeuchte;Luftdruck;

try

{

string temp = msg.Split(';')[0].Replace('.', ',');

double t = Convert.ToDouble(temp);

TempAktuellValue = new TempValue() { DateAndTime = DateTime.Now, Value = t };

double luftfeuchte = Convert.ToDouble(msg.Split(';')[1].Replace('.', ','));

FeuchteAktuellValue = new HumidValue() { DateAndTime = DateTime.Now, Value = luftfeuchte };

string d = msg.Split(';')[2].Replace('.', ',');

double druck = Convert.ToDouble(d);

PressCurrentVal = new PressureValue() { DateAndTime = DateTime.Now, Value = druck };

string ipAdr = msg.Split(';')[3];

//Daten in die Datenbank schreiben.

try

{

\_dba.InsertData(TempAktuellValue, FeuchteAktuellValue, PressCurrentVal, DateTime.Now, msg.Split(';')[3]);

}

catch (Exception ex)

{

Status = ex.Message;

}

//Max und Min-Werte feststellen und spiechern.

if (TempMaxValue != null)

{

if (TempAktuellValue.Value > TempMaxValue.Value)

{

TempMaxValue = TempAktuellValue;

}

}

else

{ TempMaxValue = TempAktuellValue; }

if (TempMinValue != null)

{

if (TempAktuellValue.Value < TempMinValue.Value)

{

TempMinValue = TempAktuellValue;

}

}

else

{

TempMinValue = TempAktuellValue;

}

//Feuchte

if (FeuchteMaxValue != null)

{

if (FeuchteAktuellValue.Value > FeuchteMaxValue.Value)

{

FeuchteMaxValue = FeuchteAktuellValue;

}

}

else

{

FeuchteMaxValue = FeuchteAktuellValue;

}

if (FeuchteMinValue != null)

{

if (FeuchteAktuellValue.Value < FeuchteMinValue.Value)

{

FeuchteMinValue = FeuchteAktuellValue;

}

}

else

{

FeuchteMinValue = FeuchteAktuellValue;

}

//Luftdruck

if (PressMaxVal != null)

{

if (PressCurrentVal.Value > PressMaxVal.Value)

{

PressMaxVal = PressCurrentVal;

}

}

else

{

PressMaxVal = PressCurrentVal;

}

if (PressMinVal != null)

{

if (PressCurrentVal.Value < PressMinVal.Value)

{

PressMinVal = PressCurrentVal;

}

}

else

{

PressMinVal = PressCurrentVal;

}

}

catch (Exception ex)

{

Status = "Fehler beim Interpretieren der Werte. " + ex.Message + Environment.NewLine + msg;

}

}

}

**1-1-2-3-1. MainViewModel.cs, zeile 159.**

public string Status

{

get

{

return \_status;

}

set

{

\_status = value;

OnPropertyChanged("Status");

}

}

**1-1-2-3-1-1. ViewModel.cs, zeile 37.**

protected virtual void OnPropertyChanged(string propertyName)

{

PropertyChanged(this, new PropertyChangedEventArgs(propertyName));

}

**1-1-2-3-1-1-1. ViewModel.cs, zeile 42.**

public event PropertyChangedEventHandler PropertyChanged;

**1-1-2-3-2. ValuesBase.cs, zeile 9.**

public class ValuesBase

{

public double Value { get; set; }

public DateTime DateAndTime { get; set; }

}

//Inheritance!!=>same default format!! => member variables(Value, DateAndTime)

//public class PressureValue : ValuesBase

//public class TempValue : ValuesBase

//public class HumidValue : ValuesBase