

Quick Start Guide



WTX Weighing API

Hottinger Baldwin Messtechnik GmbH
Im Tiefen See 45
D-64239 Darmstadt
Tel. +49 6151 803-0
Fax +49 6151 803-9100
info@hbm.com
www.hbm.com
Mat.:
DVS: HBM: public
11.2018

Hottinger Baldwin Messtechnik GmbH.

Subject to modifications.
All product descriptions are for general information only.
They are not to be understood as a guarantee of quality or
durability.

Änderungen vorbehalten.

Alle Angaben beschreiben unsere Produkte in allgemeiner Form. Sie stellen keine Beschaffenheits- oder Haltbarkeitsgarantie dar.

Sous réserve de modifications.
Les caractéristiques indiquées ne décrivent nos produits que sous une forme générale. Elles n'impliquent aucune garantie de qualité ou de durabilité.

Contents

1. System Overview

2. Prerequisites

- 2.1 Hardware
- 2.2 Software

3. Installation

- 3.1 Manual installation using the GitHub repository
- 3.2 Installation via NuGet App

4. Execute an Example Application

- 4.1 Example application „CommandLine“
- 4.2 Example application „GUIsimple“
- 4.3 Example application “GUIplc“

5. How to develop your own application

- 4.4 General structure of the Weighing API
- 4.5 Establishment of communication with the Weighing API
- 4.6 Step-by-step creation of your own application

1. System Overview

The two examples below show exemplary configurations of your system and how it can be connected via the WTX Weighing API. These configurations are illustrated in Figure 1.1.

- a) Establish a TCP/IP peer-to-peer (P2P) connection between your WTX terminal and your PC by using the RJ45 (LAN) cable, you may use an intermediary switch
- b) Establish a connection between your WTX terminal and a PC by using your wireless network

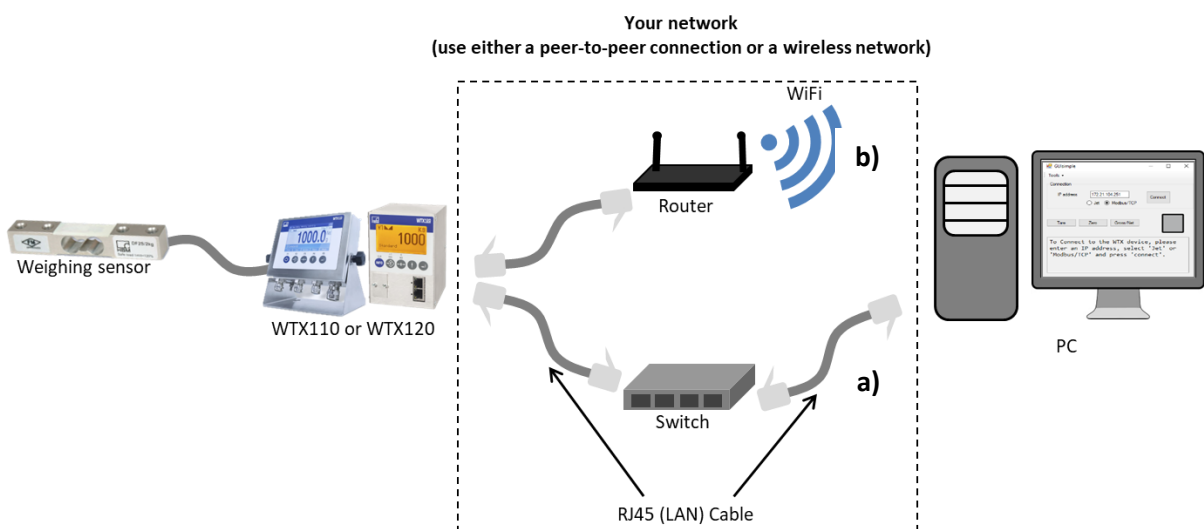


Figure 1.1: System overview

2. Prerequisites

2.1 Hardware

- HBM WTX weighing terminal (e.g. WTX110 or WTX120)
- Supply voltage for WTX110/WTX120 (12V-30V, DC)
- RJ45 Cable ("LAN" Cable)
- Weighing sensor (e.g. HBM PW6C)
- Computer with LAN-port connected to the same network as the WTX

2.2 Software

- Target operating systems:

- Windows (min. Win7 SP1, .NET Framework 4.5 or .NET Standard 2.0)
- Android (.NET Standard 2.0)
- iOS (.NET Standard 2.0)
- Programming:
 - Microsoft Visual Studio (min. VS 2012)

Visual Studio Community License – Extract:

“For organizations An unlimited number of users within an organization can use Visual Studio Community for the following scenarios: in a classroom learning environment, for academic research, or for contributing to open source projects For all other usage scenarios: In non-enterprise organizations, up to five users can use Visual Studio Community. In enterprise organizations (meaning those with >250 PCs or >\$1 Million US Dollars in annual revenue) For individuals Any individual developer can use Visual Studio Community to create their own free or paid apps.”

Download from <https://www.visualstudio.com/vs/community/> (11/28/2018) After 30 days you have to log in with a Microsoft account to unlock the test version.

- Xamarin Studio

3. Installation

This section leads you through the installation process of the WTX Weighing API.

3.1 Manual installation using the GitHub repository

- Install Visual Studio with .NET-Development extension or basic version and install required packages afterwards.
- Download the WTX Weighing API by accessing the GitHub Repository under <https://github.com/HBM/Weighing-API> and by clicking on „Clone or Download“ as indicated by the arrow in Figure 3.1.

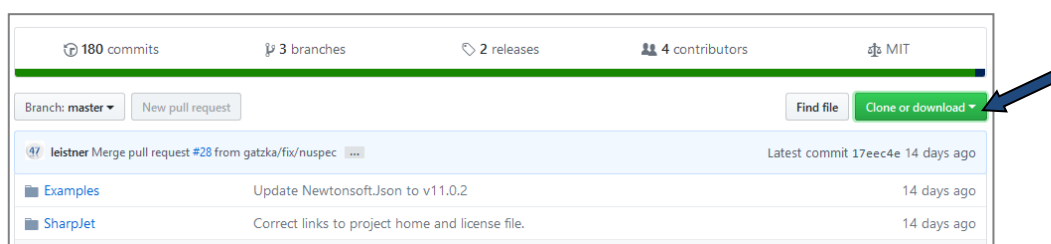


Figure 3.1: „Clone or Download“ button on GitHub to get WTX Weighing API.

- Clone the repository with git or just save and extract (right click on ZIP file) files to a local project folder on your PC.
- In the project folder, open the solution file „Weighing-API.sln“












| | | | |
|---|---------------------------------|------------------------|------|
|  | Doc | File folder | |
|  | Examples | File folder | |
|  | HBM.Weighing.API | File folder | |
|  | Tests | File folder | |
|  | .gitignore | Text Document | 1 KB |
|  | HBM.Weighing.API.vsmDI | Test Metadata File | 1 KB |
|  | LICENSE | File | 2 KB |
|  | Local.testsettings | Test Settings File | 1 KB |
|  | Readme.md | MD File | 3 KB |
|  | TraceAndTestImpact.testsettings | Test Settings File | 1 KB |
|  | Weighing-API.sln | Visual Studio Solution | 9 KB |

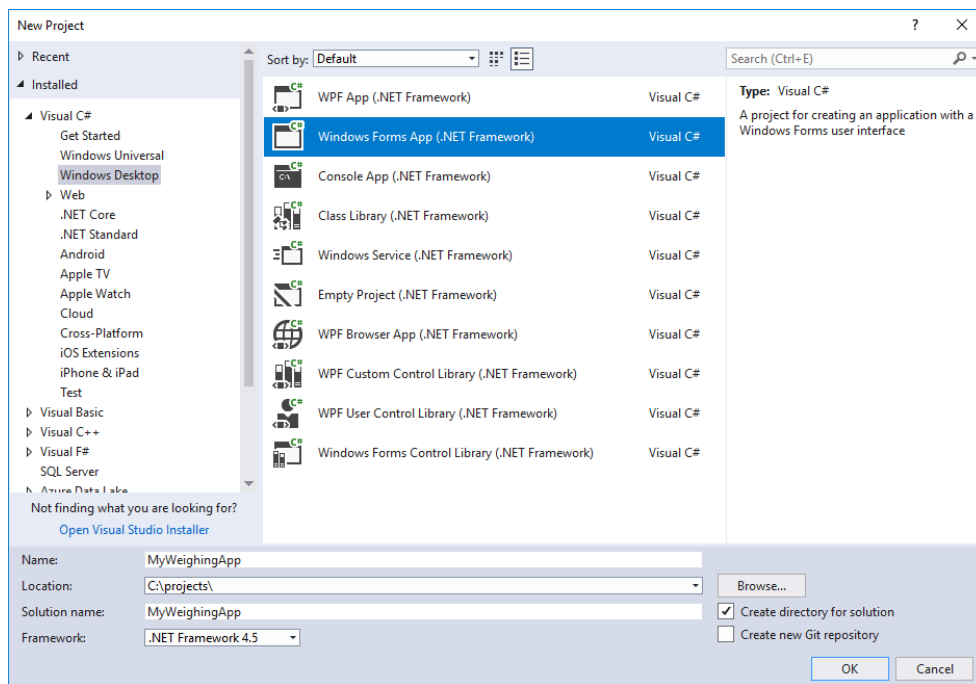
Figure 3.2: Project folder of WTX Weighing API

3.2 Installation via NuGet App

Besides the manual installation described in 3.1, it is also possible to use the NuGet deployment system, integrated in Visual Studio.

The nuget package can be found by searching for „HBM Weighing API“ in the NuGet browser, as described in fogure 3.4.

- Create a new Visual Studio project and select Windows Forms App



- Right click on the MyWeighingApp solution in the Visual Studio Project Explorer, then click on „Manage NuGet Packages...” in the drop-down menu as shown in Figure 3.3.

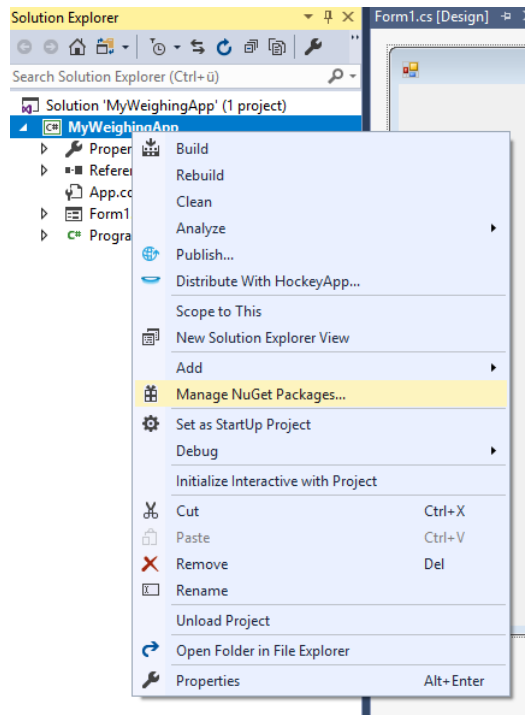


Figure 3.3: Drop-down menu in Visual Studio Solution Explorer

- Click on „Browse“ in the upper left corner of the appearing window and search for „HBM.Weighing.API“ like in Figure 3.4.
(Until HBM.Weighing.API is officially released you have to check “Include prerelease”!)

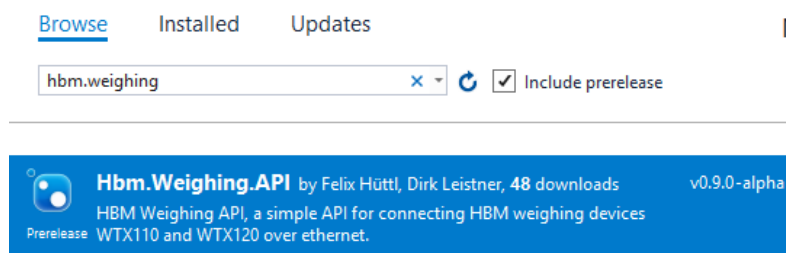


Figure 3.4: NuGet Package Manager

- Click on „Install“ on the right side of the appearing subwindow, see Figure 3.5.

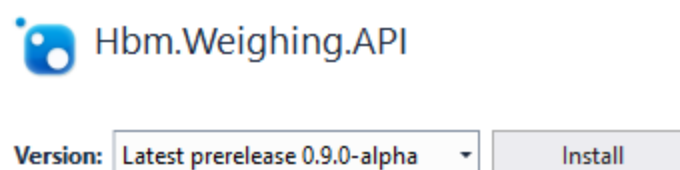


Figure 3.5: Installation button for current NuGet application

4. Execute an Example Application

In addition to the HBM.Weighing.API itself, three sample applications are available in the Github repository:

- A console application „CommandLine“
- A Windows Form GUI application „GUIplc“
App with all input and output words for your WTX terminal for Modbus.
- A Windows Form GUI application „GUIsimple“
A quite simple GUI for demonstrating the most important use cases.

To be able to execute one of the three provided example applications, the WTX Weighing API has to be installed as described in section 3.1. Furthermore, your WTX terminal has to be connected with the same network as your PC.

How to execute one of the example applications and their functionality is described in the following subsections.

4.1 Example Application „Command Line“

The command line application enables you to establish a Modbus/TCP connection or a Jetbus connection if you use a WTX120 weighing terminal. With a WTX110 weighing terminal, only a Jetbus connection can be established. The application provides values such as the current net or gross weight in the command line of your system. All values are updated event based, i.e. as the measured value changes.

How to execute the CommandLine application:

- To execute the „CommandLine“ application, right click on the project “CommandLine” in the Solution Explorer and then click on „Properties...” .
- Select „Debugging“ in the appearing window.
- Type „Modbus“ and the IP address of your WTX terminal in the arguments box to establish a Modbus TCP connection. If you want to establish a Jetbus connection, just type „Jet“ instead of „Modbus“.

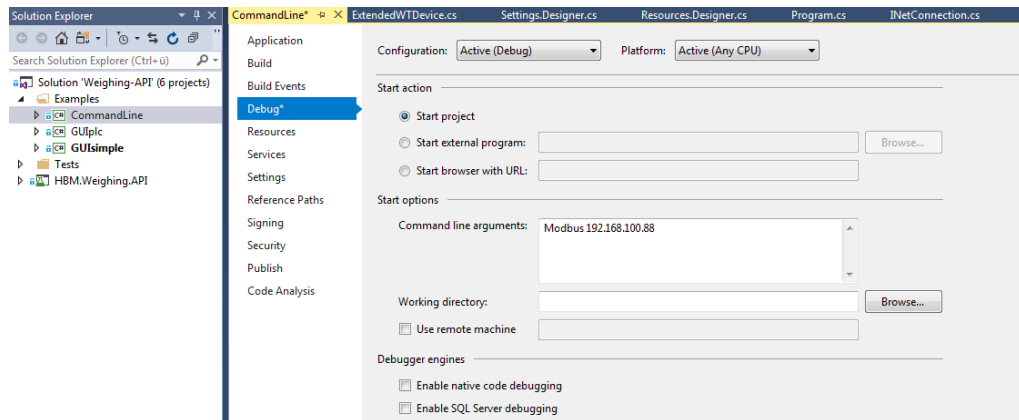


Figure 4.1: Necessary command line arguments to connect to WTX

- Now compile and run the project CommandLine
- If the application has been built and executed successfully, the window shown in Figure 4.2 appears. All relevant values of your connected WTX terminal are displayed here.

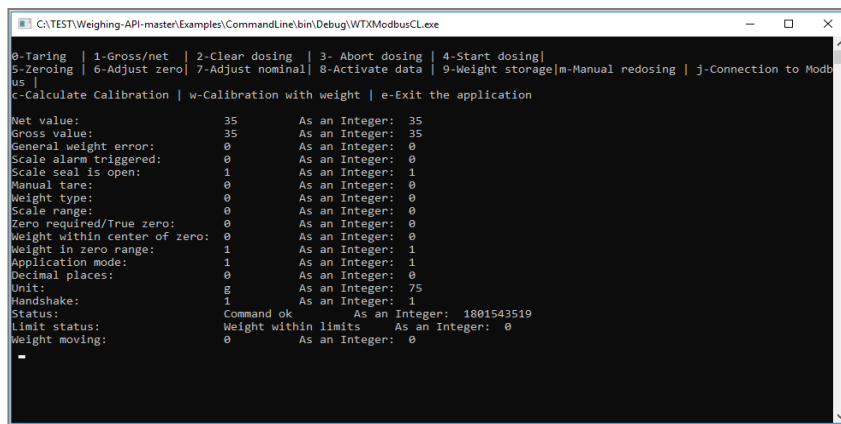


Figure 4.2: CommandLine application window.

4.2 Example Application „GUIsimple“

The GUIsimple application enables you to establish a Modbus/TCP connection or a Jetbus connection if you use a WTX120 weighing terminal. With a WTX110 terminal, only a Jetbus connection can be established.

How to execute the GUIsimple application:

- To execute the „GUIsimple“ application just choose the corresponding name in the drop down menu shown in Figure 4.4 and click on „Start“.

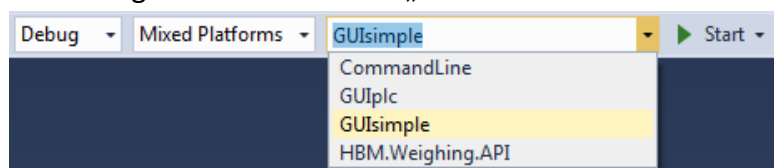


Figure 4.4: Drop down menu to choose between projects

- In case of a successful execution, the windows shown in Figure 4.5 appears.

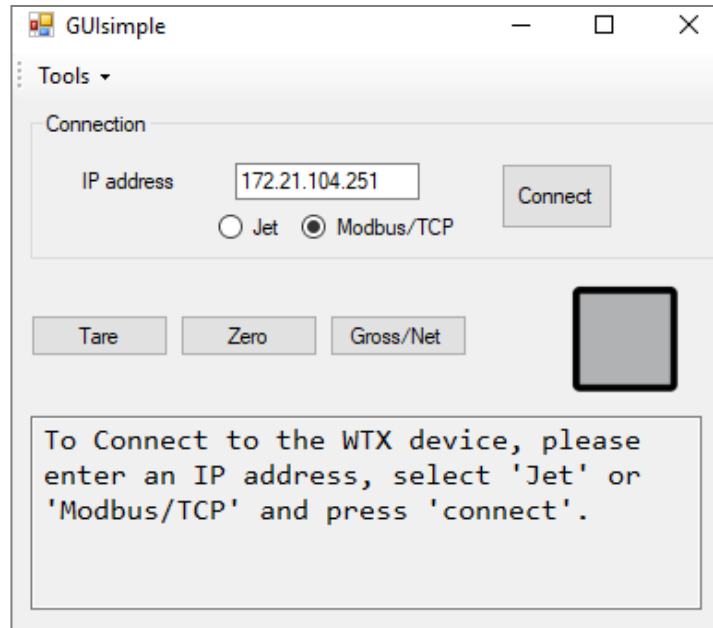


Figure 4.5: GUIsimple application window before connect

- Now type in the IP address of your WTX terminal, choose Modbus/TCP or Jet and click Connect. This may take some seconds. If the connection has been established successfully, the window shown in Figure 4.6 appears.

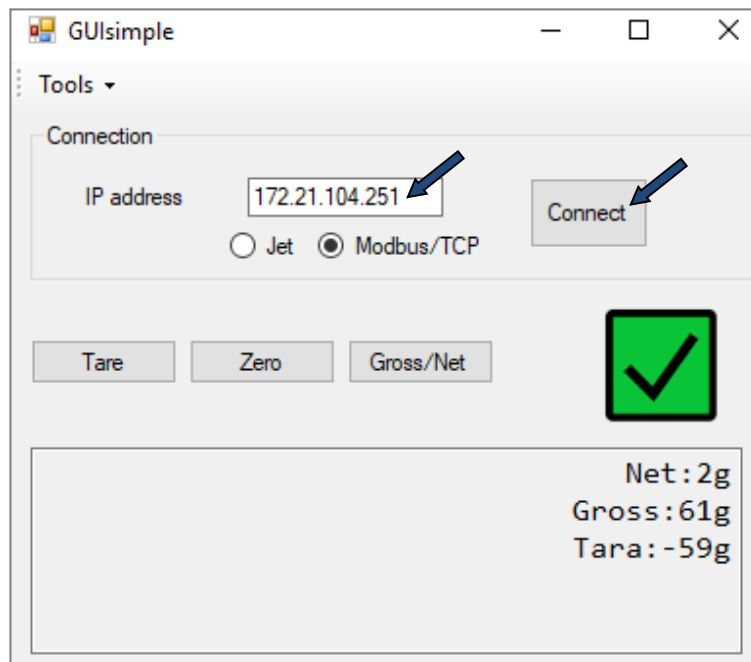


Figure 4.6: GUIsimple application window after connect.

- The current weight value provided by the connected WTX terminal is displayed in the GUIsimple window. By clicking on Gross/Net it can be switched between the Gross and net value. Furthermore, the zeroing and tare functions of your WTX terminal can be triggered by clicking on the corresponding buttons.

4.3 Example Application „GUIplc“

The GUIplc application enables you to establish a Modbus/TCP connection with a WTX120 weighing terminal. With this application one can access all possible functions provided by the WTX terminal via Modbus/TCP.

- The example application „GUIplc“ can be opened in the same way as the „CommandLine“ application.
- The GUIplc application window is shown in Figure 4.7

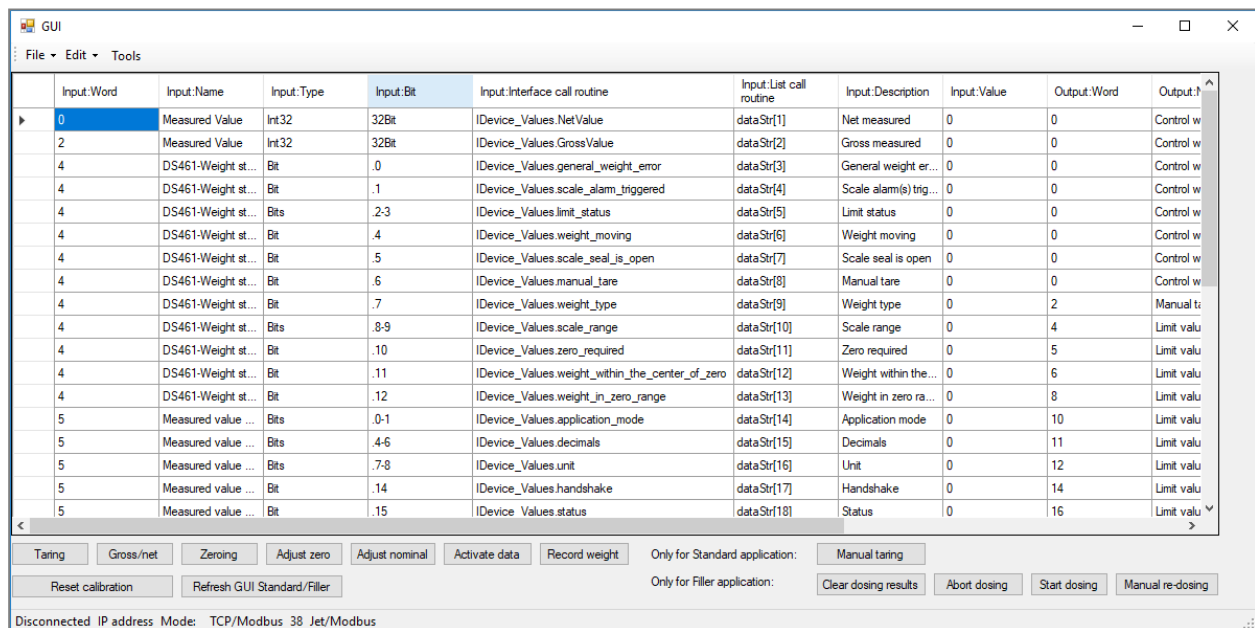


Figure 4.7: GUIplc application window

5. Develop your own Application

One major benefit of the WTX API is that it enables you to easily develop your own weighing application, which perfectly matches your personal requirements. Therefore, you can just adapt one of the example applications presented in section 3 or you completely develop a new application from scratch. This section contains a general UML overview of the WTX Weighing API code structure (subsection 5.1) and short description of the communication flow between the different classes and interfaces (subsection 5.2). A step-by-step guide for the creation of an exemplary application comprising basic functions, is provided in subsection 5.3.

5.1 General Structure of the Weighing API

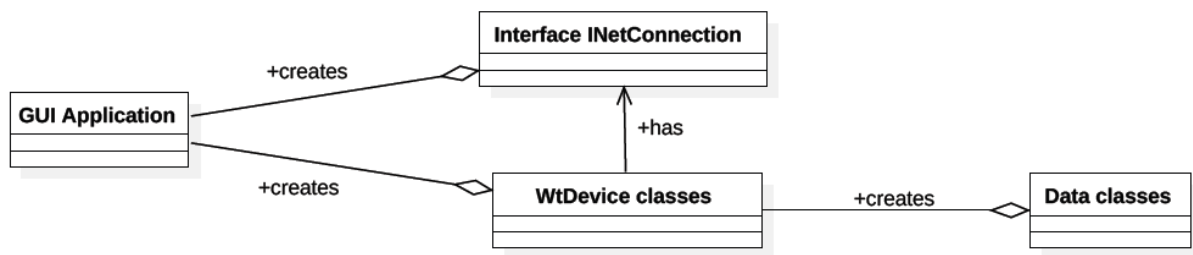


Figure 5.1: Simplified UML class diagram of WTX Weighing API

The general API structure is developed according to the three-tier architecture. There is a presentation tier/layer which is the top-most level. Users can access and communicate directly to it. The 'GUI Application' represents it. It translates and shows tasks, input and output from the lower layers in a user-friendly way. You can read the measured value, tare and calibrate the device for example.

There is a process logic tier containing the data processing and computing. It coordinates the application, processes logical decisions, commands, controls the communication and performs calculations. It transfers data from the data layer to the presentation layer. The classes between the data and presentation tier represent the process logic tier (Connection interface and classes, WtDevice classes).

There is a data storage tier, which is the lowest level. It contains the data, its storage and update. It is represented by the data classes and its interfaces (ProcessData, DataStandard, DataFiller, DataFillerExtended).

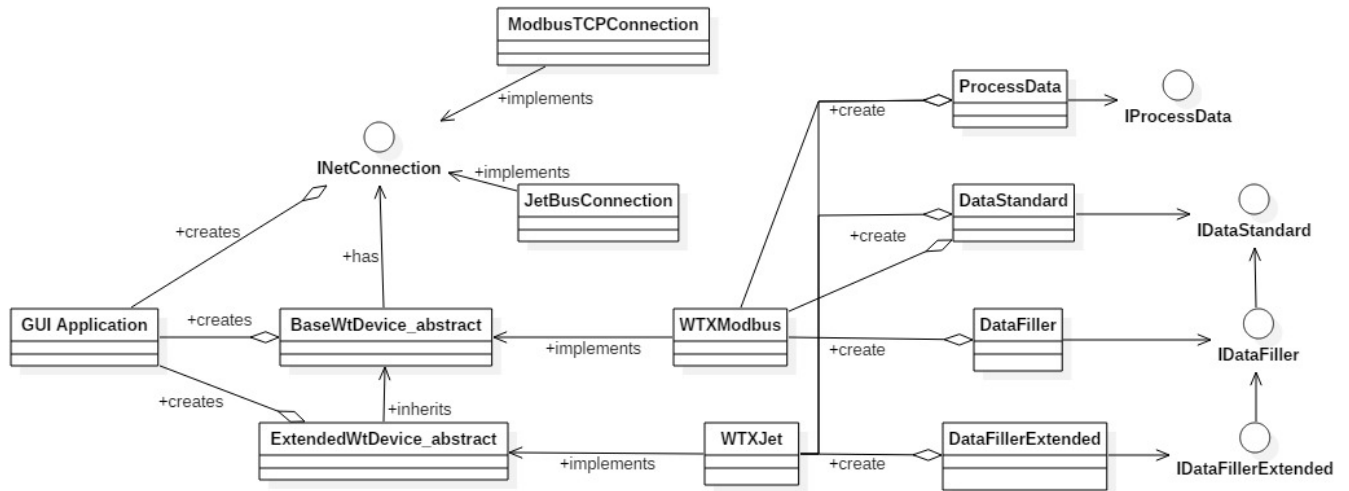


Figure 5.2: Full UML class diagram of WTX Weighing API

The interface `INetConnection` is implemented by the classes `ModbusConnection` and `JetBusConnection`. An object of the connection class and `WtDevice` class is necessary to run the API.

The `WtDevice` classes are split to several classes : The abstract class `BaseWtDevice` is implemented by `WtxModbus` class containing basic process functions and methods which are available for every connection type (for Modbus and Jetbus). As an extension Jetbus provides more functions and data compared to Modbus, so there is an `ExtendedWTDevice` (abstract class) implemented by the `WtxJet` class.

`WTXModbus` holds an instance of `DataStandard` or `DataFiller` (for filler mode). `DataFillerExtended` is held by the class `WTXJet`, because additional data is available with a Jetbus connection. `ProcessData`, `DataStandard` contains data, which can be used by Modbus and Jetbus connection. `ProcessData` defines the properties (attributes) for periodically updated values, which are continuously displayed in the application after a timer interval. It contains essential process data like weight value, unit, status etc. .

To get the Weighing Api started you need to create objects of `ModbusTcpConnection` and `WtxModbus` or `JetBusConnection` and `WtxJet` as explained in the following section.

5.2 Establishment of communication with Weighing API

The following steps are required for synchronous and/or asynchronous data transfer:

1. Create `INetConnection`

An object of class `ModbusConnection` or `JetBusConnection` has to be created first, in order to establish a Modbus or JetBus connection respectively. Both constructors require the IP address of the WTX terminal which is intended to get connected

2. Create WtDevice

Create an object of `WtxModbus` or `WtxJet` for a Modbus or JetBus connection respectively. The constructor requires an instance of the corresponding connection class (`ModbusConnection` or `JetBusConnection`) and a timer interval.

3. Connect device

Simply call the `Connect()` function on the `WtDevice` instance.

4. Get data synchronously

After connecting to a `WtDevice`, get or set any data directly from the `WtDevice` instance.

```
ModbusConnection _modConn = new ModbusConnection(ipAddress); //1.
WtxModbus device = new WtxModbus(_modConn, timerIval); //2.

device.Connect(); //3.
int netWeight = device.NetValue;
```

Figure 5.3: Code example for establishment of synchronous connection

The code fragment in Figure 5.3 shows a simple example for a synchronous data transfer with a WTX device to read the current net weight value.

5. Get process data asynchronously

If an asynchronous data transfer is desired, e.g. for continuously updating the weight net value from the example shown in Figure 5.3, an additional callback function can be passed to the constructor of `WtxModbus` or `WtxJet`.

```

ModbusConnection _modConn = new ModbusConnection(ipAddress); //1.
WtxModbus device = new WtxModbus(_modConn, timerIval, updateCallback); //2.
device.Connect(); //3.
int netWeight = device.NetValue; //4.

//5.
private void updateCallback(object sender, ProcessDataReceivedEventArgs e)
{
    this.BeginInvoke(new Action(()=>
    {
        int netWeight = e.ProcessData.NetValue;
    }));
}

```

Figure 5.4: Code example for establishment of asynchronous connection

Figure 5.4 provides a code example for the establishment of an asynchronous data transfer to continuously update the net weight value, whereas the update interval depends on the parameter 'timerIval' defined in (2.). Parameter 'e' is of type ProcessDataReceivedEventArgs, and contains all the process data.

5.3 Step-by-step creation of your own application

This section aims to provide you a guide for developing your own app. Therefore, the development of a simple GUI application is presented step-by-step. The final application is similar to the „GUIsimple“ application from subsection 4.2.

Create a New Project

In order to develop your own HBM API Weighing application, please first open Visual Studio and add a new project as shown in Figure 5.5.

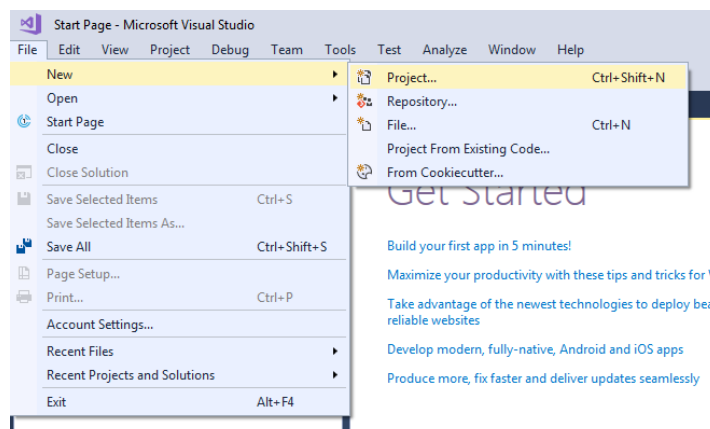


Figure 5.5: Creation of new project

Thereupon, the window shown in Figure 5.6 appears in which you can choose the project type. Choose e.g. „Windows Forms App (.NET Framework)“ and name the application.

In general, you can also choose other project types, such as Console-App, but for our example app we focus on Windows-Forms.

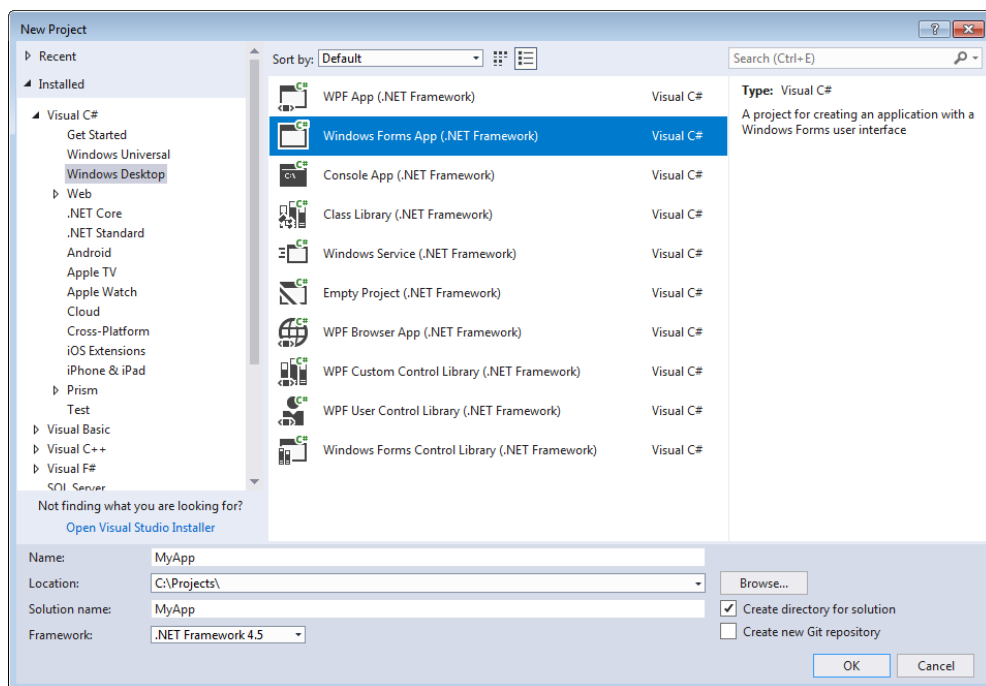


Figure 5.6: Properties of new windows forms application

After clicking OK, the empty design window, see Figure 5.7, for the application appears.

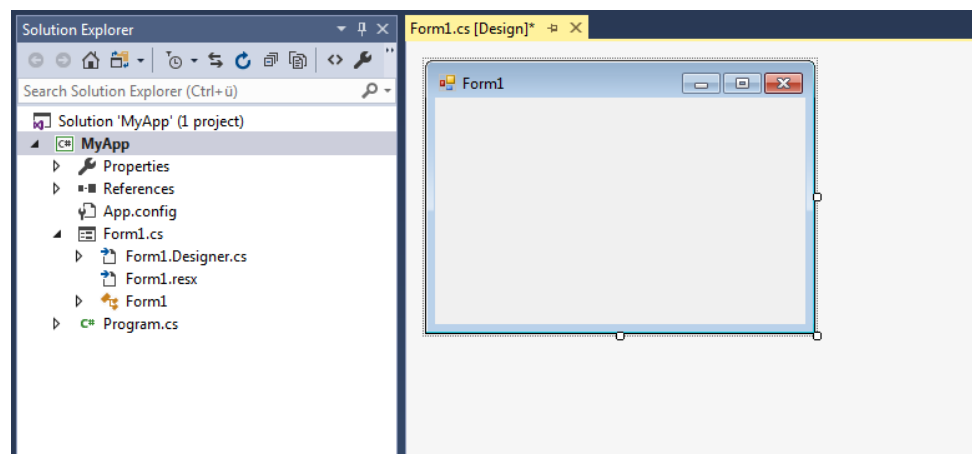


Figure 5.7: Empty design window of application

Graphical editing of GUI window

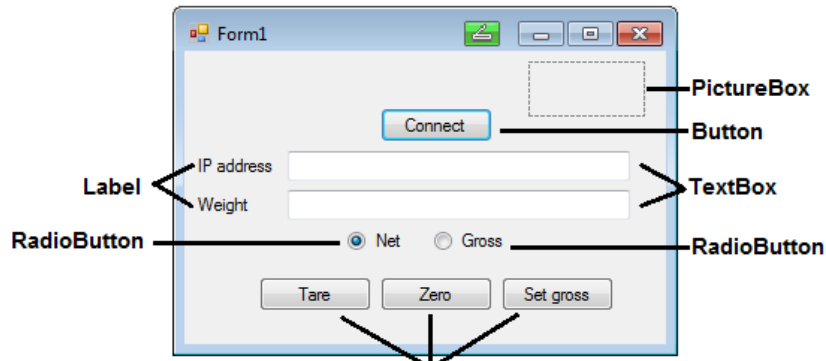
The application window can easily be modified via Drag & Drop, using the items provided by the toolbox. For the set up of our „MyApp“ application, five different item types are utilized: TextBox, Button, RadioButton, Label and PictureBox.

Figure 5.8: Form design window of the examplay application

Therewith, the graphical part of our individual application is almost done.

For better code visibility, please change the „Name“ attribute of each item according to the naming scheme given by Table 5.1. This naming scheme is used in the further course of this guide.

| Item | Text | Name |
|--------|---------|---------------|
| Button | Connect | connectButton |



| | | |
|--------------|----------------|------------------|
| | Tare | tareButton |
| | Zero | zeroButton |
| Radio Button | Net | netRadioButton |
| | Gross | grossRadioButton |
| Text Box | - | ipTextBox |
| | - | weightTextBox |
| Label | Individual App | labelTitle |
| | IP Address | labelIp |
| | Weight | labelWeight |
| PictureBox | - | logoPictureBox |

Table 5.1: Naming of all items used for the IndividualApp

Install HBM.Weighing.API via Nuget

Right-click your solution and choose “Manage NuGet Packages for Solution...”.

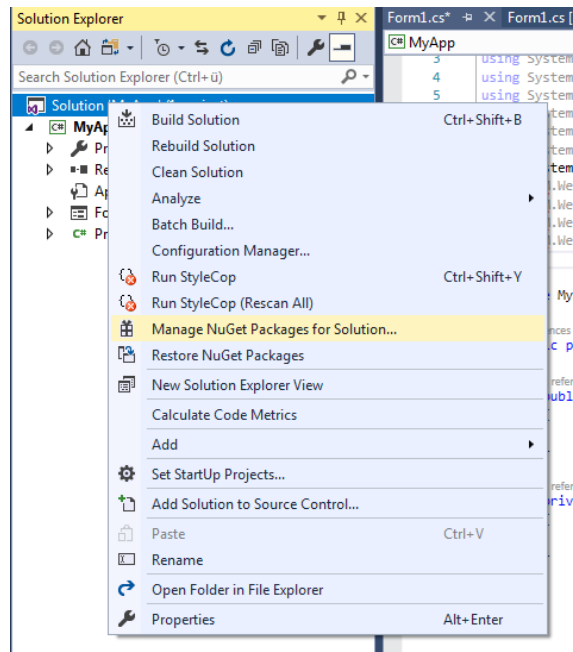


Figure 5.9: Install the HBM.Weighing.API

Now you find the nuget package by filtering the Browse tab e.g. with “HBM”.

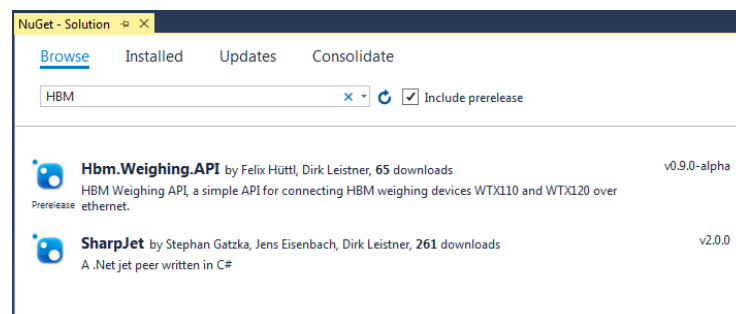


Figure 5.10: Locate the nuget package HBM.Weighing.API in the nuget.org repository

**Note to check the “Include prerelease” checkbox, while HBM.Weighing.API is not officially released!*

Select the package Hbm.Weighing.API and install it to your application as shown in figure 5.11.

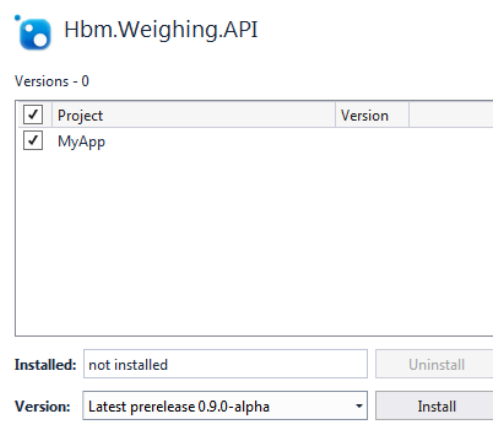


Figure 5.11: Install the nuget package

Editing of the program code

Please open the program code, by double clicking on an arbitrary point in the MyApp window. Thereby, code in Figure 5.9 appears.

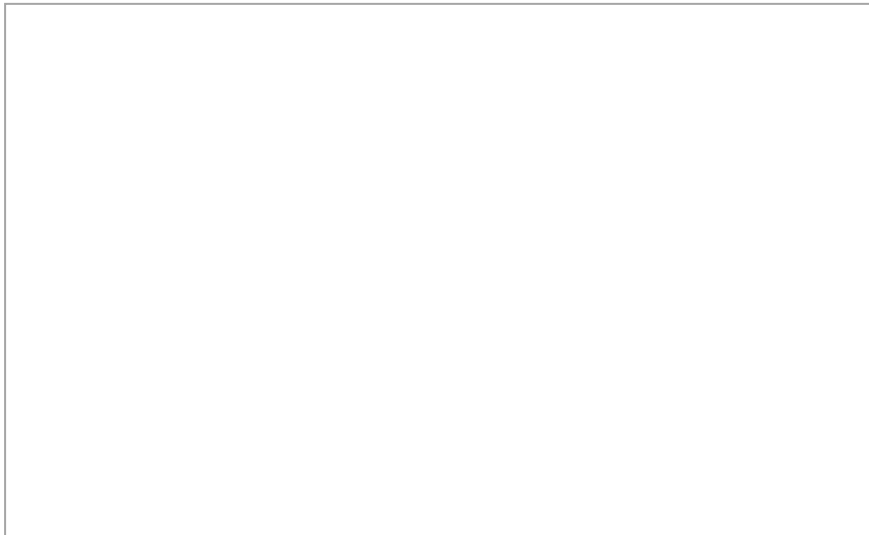


Figure 5.12: Initial code for MyApp

Note: It might be necessary to create a reference to the here included libraries by right clicking on the error message next to one of the here included directives.

In this exemplary implementation, only a JetBus connection is established, whereas the connection via Modbus/TCP is very similar. Please declare an object of type JetBusConnection and of type WtxJet, which implements the abstract classes BaseWtDevice and ExtendedWtDevice . The connection itself is established by using the connect function of the WtxJet class. It needs an instance of JetBusConnection, a timer interval in ms and an EventHandler as callback function as update. A timer interval of 2000 ms is used here. The connect function has to be executed within a try-catch frame because different exceptions can be thrown during its execution.

The resulting code fragment is shown in Figure 5.11. A JetBusConnection constructor requires the IP address of your device as input argument, which is obtained from the IP address text box using the class attribute IP Address. This JetBusConnection object serves as an input parameter for the WtxJet constructor. The second parameter “timerlval” is the timer interval for the continuous reading of the Wtx device. The third parameter „update“ is a event handler, a function called once the event ProcessDataReceived in a WtDevice class is raised after the wtx device is read out. It is introduced in the following section. Message boxes are used to show whether a connection has been established successfully after clicking the connect button or not.

```

namespace MyApp
{
    public partial class Form1: Form
    {
        WtxJet _wtxDevice;

        public Form1()
        {
            InitializeComponent();
        }

        private void connectButton_Click(object sender, EventArgs e)
        {
            int _timerIval=200;
            string _ipAddress = ipTextBox.Text;
            JetBusConnection _jetConnection = new JetBusConnection(_ipAddress);
            _wtxDevice = new WtxJet(_jetConnection, _timerIval, update);

            try
            {
                _wtxDevice.Connect(2000);
                MessageBox.Show("Connection successful");
            }
            catch (Exception)
            {
                MessageBox.Show("Connection failed");
            }
        }
    }
}

```

Figure 5.10: Implemented connectButton handler function

In order to be able to continuously read the weight value of your WTX terminal, a callback function `update(object sender, ProcessDataReceivedEventArgs e)` has to be added first. This function is shown in Figure 5.12. The weight value variable is assigned depending onto which radio button (Net and Gross) is checked. Variable `decimal`, which is implemented in the function header together with the weight variable, is used to determine the displayed number of digits. The weight is displayed by assigning the value of the weight variable to the weight text box.

`BeginInvoke` initiates a thread, an execution path, asynchronously and is used to update the weight value and to display overload or underload here.

It is necessary because of using asynchronous reading and writing you have several threads. So, you have to initiate one thread explicitly by using `BeginInvoke` otherwise you get an exception failure.

```
private void update(object sender, ProcessDataReceivedEventArgs e)
{
    int weight;
    int decimals = e.ProcessData.Decimals;

    weightTextBox.BeginInvoke(new Action(() =>
    {
        if (this.netRadioButton.Checked)
            weight = e.ProcessData.NetValue;
        else
            weight = e.ProcessData.GrossValue;

        weightTextBox.Text = _wtxDDevice.CurrentWeight(weight, decimals)
            + " " + _wtxDDevice.Unit;
    }));

    weightTextBox.BeginInvoke(new Action(() =>
    {
        if (e.ProcessData.Underload == true)
        {
            weightTextBox.Text = "Underload";
        }
        if (e.ProcessData.Overload == true)
        {
            weightTextBox.Text = "Overload";
        }
    }));
}
}
```

Figure 5.12: Code to asynchronously read net or gross value

In the next step, the tare and zero button handler functions are implemented. As already done for the connect button, the tare and zero button handler functions can be added by double clicking on the corresponding button in the application design window. These handlers can now easily be implemented by using the zero and tare function of BaseWtDevice (by its derived class WtxJet), as shown in Figure 5.13.

```
...
private void tareButton_Click(object sender, EventArgs e)
{
    _wtxDDevice.Tare();
}

private void zeroButton_Click(object sender, EventArgs e)
{
    _wtxDDevice.Zero();
}
}
```

Figure 5.13: Implementation of taring and zeroing function

The application can now be tested already by clicking on „Run“ or by pressing „F5“. After a successful build of your code, a window as shown in Figure 5.14. will appear. Make sure that your WTX terminal is connected and enter its IP address to the corresponding textbox and click on „Connect“. A message box stating „Connection establishment has been successful“ appears on your screen in case of a successful connection establishment, see Figure 5.14. Otherwise, the message „Connection establishment failed“ will appear. If the connection failed, please check if your WTX terminal is properly connected in accordance with the WTX Quick Start Guide and if you entered the correct IP address of your WTX terminal.

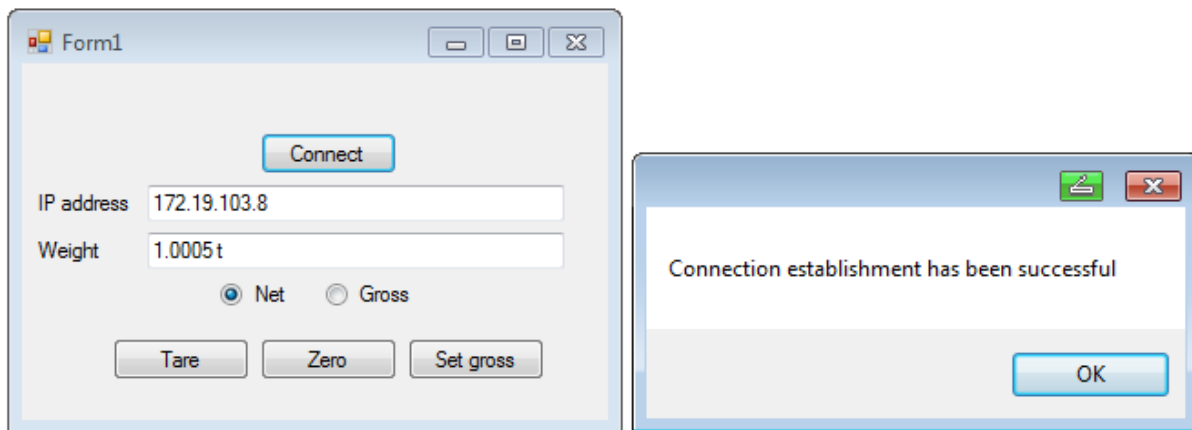
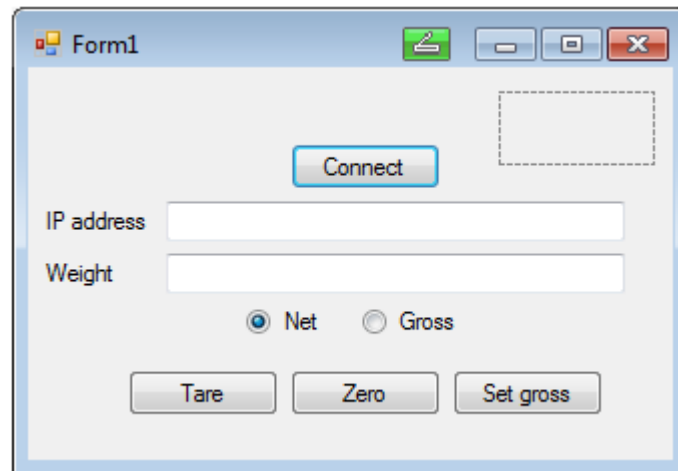


Figure 5.14: Established connection successfully

If the connection is established successfully, the actually measured weight value of your WTX device is also displayed in the weight text box.

In the next step, the tare and zero button handler functions could be implemented. As already done regarding the connect button, the tare and zero button handler functions can be added by double clicking on the corresponding button in the application design window. These handler functions can now easily be implemented by using the zero and tare function of the class 'BaseWtDevice', as shown in Figure 5.13. The tare and zero buttons can be used already as well.

Finally, we use the 'PictureBox' which has been added during the window design, giving this exemplary application its final touch. In order to choose your individual logo just go to the windows forms design window and click on the picture box. By clicking on „Choose picture...“ in the lower right corner you can now insert any picture. For this example, the HBM logo has been used. The final application is presented in Figure 5.15.



5.15: Complete executed application

The entire code of the application code is shown on the next two pages.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using HBM.Weighing.API;
using HBM.Weighing.API.WTX;
using HBM.Weighing.API.WTX.Jet;
using HBM.Weighing.API.WTX.Modbus;

namespace IndividualApp
{
    public partial class IndividualApp : Form
    {
        private string _ipAddress;
        private JetBusConnection _jetConnection;
        private WtxJet _wtxDevice;
        private int _timerInterval;

        public IndividualApp()
        {
            _timerInterval = 200;
            InitializeComponent();
        }

        private void connectButton_Click(object sender, EventArgs e)
        {
            _ipAddress = ipTextBox.Text;
            _jetConnection = new JetBusConnection(_ipAddress, "Administrator", "wtx");
            _wtxDevice = new WTXJet(_jetConnection, _timerInterval, update);

            try
            {
                _wtxDevice.Connect(2000);
                MessageBox.Show("Connection establishment has been successful");
            }
            catch (Exception)
            {
                MessageBox.Show("Connection establishment failed");
            }

            _timerInterval,      }

        ...
        ...// see on next page
    
```

```

...

private void update(object sender, ProcessDataReceivedEventArgs e)
{
    string weightString = "0.0";
    double dWeightNet = 0;
    double dWeightGross = 0;

    weightTextBox.BeginInvoke(new Action(() =>
    {
        if (this.netRadioButton.Checked)
        {
            weightString = _wtxDDevice.PrintableWeight.Net;
            dWeightNet = _wtxDDevice.Weight.Net;
            dWeightNet = e.ProcessData.Weight.Net;
        }
        if (this.grossRadioButton.Checked)
        {
            weightString = _wtxDDevice.PrintableWeight.Gross;
            dWeightGross = _wtxDDevice.Weight.Gross;
            dWeightGross = e.ProcessData.Weight.Gross;
        }
        weightTextBox.Text = weightString + " " + _wtxDDevice.Unit;
    }));

    weightTextBox.BeginInvoke(new Action(() =>
    {
        if (e.ProcessData.Underload == true)
        {
            weightTextBox.Text = "Underload";
        }
        if (e.ProcessData.Overload == true)
        {
            weightTextBox.Text = "Overload";
        }
    }));
}

private void tareButton_Click(object sender, EventArgs e)
{
    _wtxDDevice.Tare();
}

private void zeroButton_Click(object sender, EventArgs e)
{
    _wtxDDevice.Zero();
}

private void setGrossButton_Click(object sender, EventArgs e)
{
    _wtxDDevice.SetGross();
}

}

```