

Operating Manual | Bedienungsanleitung

English

Deutsch

HBM Automation API

API for Industrial Electronics



Hottinger Brüel & Kjaer GmbH
Im Tiefen See 45
D-64293 Darmstadt
Tel. +49 6151 803-0
Fax +49 6151 803-9100
info@hbm.com
www.hbm.com

Mat.:
DVS: A05600_01_X00_00 HBM: public
11.2020

© 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 Haltbarkeits-
garantie dar.

Operating Manual | Bedienungsanleitung

English

Deutsch

HBM Automation API

API for Industrial Electronics



1	Safety Instructions	3
2	Markings used	4
3	Overview	5
3.1	System description	5
3.2	Operating requirements	6
3.2.1	Supported devices	6
3.2.2	Required hardware	6
3.2.3	Required software	7
3.3	Connecting to a PC	7
4	Installation	8
4.1	Installing the data for the sample applications	8
4.2	Installing the HBM Automation API for your own projects	10
5	Operating principle and functions of the HBM Automation API	12
5.1	General structure of the HBM Automation API	12
5.2	Basic functions in BaseWTDevice	14
6	Examples	17
6.1	Example CommandLine	17
6.2	Example GUIsimple	19
6.3	Example GUIplc	21
7	Developing custom applications	23
7.1	Communicating via the HBM Automation API	23
7.2	Creating your own custom application step-by-step	24
7.2.1	Creating a new project	24
7.2.2	Designing the GUI	26
7.2.3	Writing program code	28
8	FAQs	36
9	Technical Support	38

1 Safety Instructions


Intended use

The HBM Automation API may only be used in conjunction with HBM devices listed in this manual. Use for any purpose other than the above is deemed improper use.

The HBM Automation API provides the device functions for the Visual Studio software program. The device operating manuals contain the safety instructions to be observed when operating the devices.

2 Markings used

Important instructions are specifically identified. Following these instructions is essential in order to prevent accidents and damage to property.

Icon	Meaning
 Information	This marking draws your attention to information about the product or about handling the product.
<i>Emphasis</i> <i>See ...</i>	Italics are used to emphasize and highlight text and identify references to sections of the manual, diagrams, or external documents and files.
Device → New	Bold text indicates menu items, as well as dialog and window headings in the program environment. Arrows between menu items indicate the sequence in which the menus and sub-menus are called up
<i>Sample rate</i>	Bold text in italics indicates inputs and input fields in the user interfaces.

3 Overview

Make sure that you always use the operating manual version applicable to your device, and the latest version of the relevant firmware. The latest version can always be found on the HBM website at <https://www.hbm.com/Downloads>.

See also *chapter 8, page 36*, where some of the terms used here are explained.

3.1 System description

The HBM Automation API (Application Programming Interface) is an open-source interface that allows you to access the functionality of various devices, including WTX weighing terminals, via Ethernet (TCP/IP). This enables you to quickly create your own application programs optimally adapted to your applications. The API makes this process easier by providing predefined classes, methods and variables that you can embed directly into the Visual Studio development environment. So the details – that is, which commands and data in what combination have to be sent to the devices – are not important. The API also gives you the ability to log, process, share and view data from multiple sources. In addition, the API contains freely available code samples as templates which you can modify for your specific application and requirements.

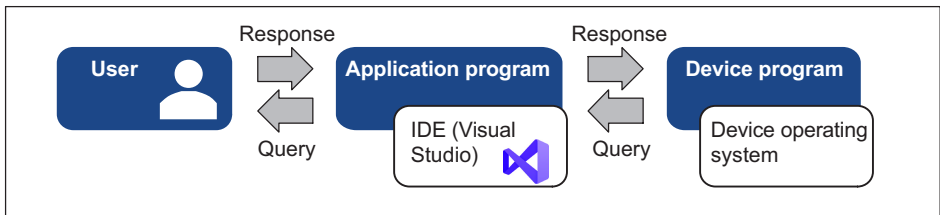


Fig. 3.1 How the HBM Automation API works

You can connect to the devices via Ethernet TCP/IP by the JET bus or alternatively to some devices, including the WTX120, via the Modbus interface of your control system.

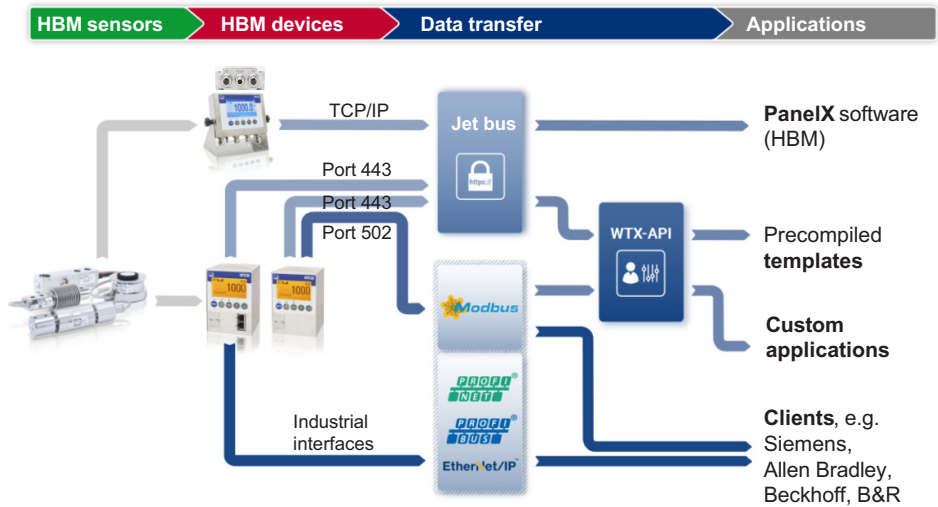


Fig. 3.2 Examples of the HBM Automation API in the context of the measurement chain, devices and transducers are shown.

3.2 Operating requirements

You need specific hardware and software in order to use the HBM Automation API.

3.2.1 Supported devices

- Weighing terminal WTX110
- Weighing terminal WTX120
- Digital sensor electronics DSE-HIE

3.2.2 Required hardware

- One of the supported devices.
- Power supply for the device, e.g. 12 V ... 30 V_{DC} for the WTX devices or 15 V ... 30 V_{DC} for the DSE.

- Transducer connected to the HBM device.
- PC with LAN or WLAN connection to the same network as the device.
- For Visual Studio, your PC should have approximately 50 GB of available hard disk space, 4 GB of RAM, and a 2 GHz CPU with at least 2 cores.

3.2.3 Required software

- Windows operating system, min. Windows® 7 SP1.
- Visual Studio, min. version 2013.
- .Net, min. version 4.5.2.

3.3 Connecting to a PC

There are two ways to set up a connection between a HBM device and PC:

1. Connect via WLAN.
2. Make an Ethernet connection via TCP/IP. Either by a direct peer-to-peer (P2P) connection or by a switch.

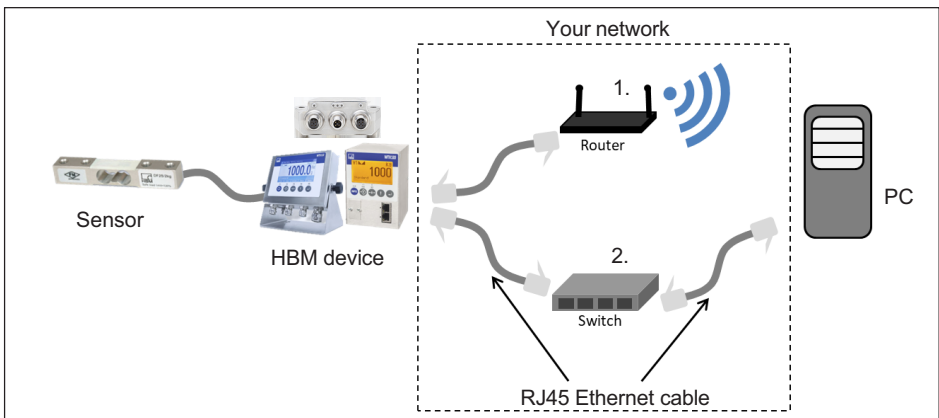


Fig. 3.3 System view with connection options

4 Installation

For all applications you need the (free) HBM Automation API (Hbm.Automation.Api), which you integrate into Visual Studio using the NuGet package manager – see *section 4.2*. To help you get started, HBM provides three executable sample program applications on GitHub (also available free of charge) (see *section 4.1*):

- The "CommandLine" console application.
- The Windows Forms "GUIplc" application, which includes all the inputs and outputs for a WTX weighing terminal.
- The Windows Forms "GUIsimple" application, which includes only the most important functions.

To run the samples, you need to install the .NET desktop development in Visual Studio (on the **Workloads** tab of the Visual Studio Installer).

4.1 Installing the data for the sample applications

To download all the data, you first need to install some components in Visual Studio:

- ▶ Start the Visual Studio Installer.
- ▶ Click on **Change**.
- ▶ Choose the **Individual components** tab.
- ▶ In the **Code tools** section, choose the **Git for Windows** and **GitHub extension for Visual Studio** components – see *Fig. 4.1*.
- ▶ If it is not already installed, also choose the **NuGet Package Manager** (*Fig. 4.1*).
- ▶ To install, click on **Change**.

The components are then downloaded and installed.

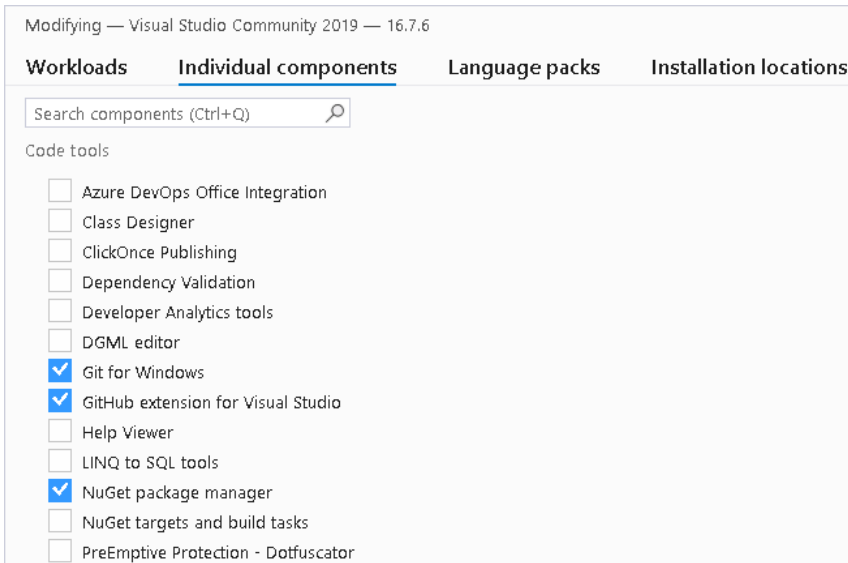


Fig. 4.1 Installing the required components

Using these components, in the next step you can download the sample files from GitHub.

- ▶ Start Visual Studio.
- ▶ Click on **Clone a repository**.
- ▶ Set **<https://github.com/HBM/Automation-API>** as the repository location – see Fig. 4.2.
- ▶ Also specify the local path where you want to store the project ("Source\Repos\Automation-API" folder in Fig. 4.2).
- ▶ Click on **Clone**.

The data is downloaded and the project is opened. You will find the sample descriptions in *chapter 6*.

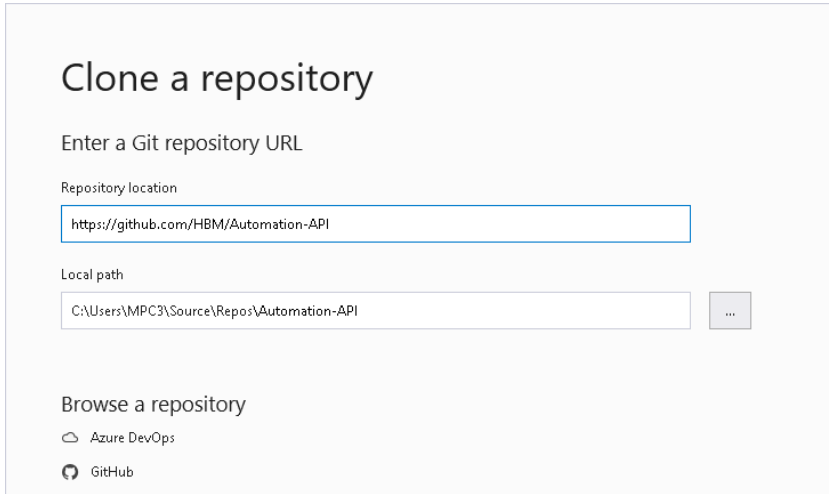


Fig. 4.2 Specifying the repository for the sample applications

4.2 Installing the HBM Automation API for your own projects

If you only need the API for your own programs, and the NuGet package manager is already installed (if not see *section 4.1*), do the following:

- ▶ Start Visual Studio and create your project as usual.
- ▶ Choose **Project** → **Manage NuGet Packages** to run the package manager.
- ▶ Look for ***Hbm.Automation.Api***.
- ▶ Select the entry and click ***Install*** (Fig. 4.3).

The HBM Automation API is downloaded and is available in your project.

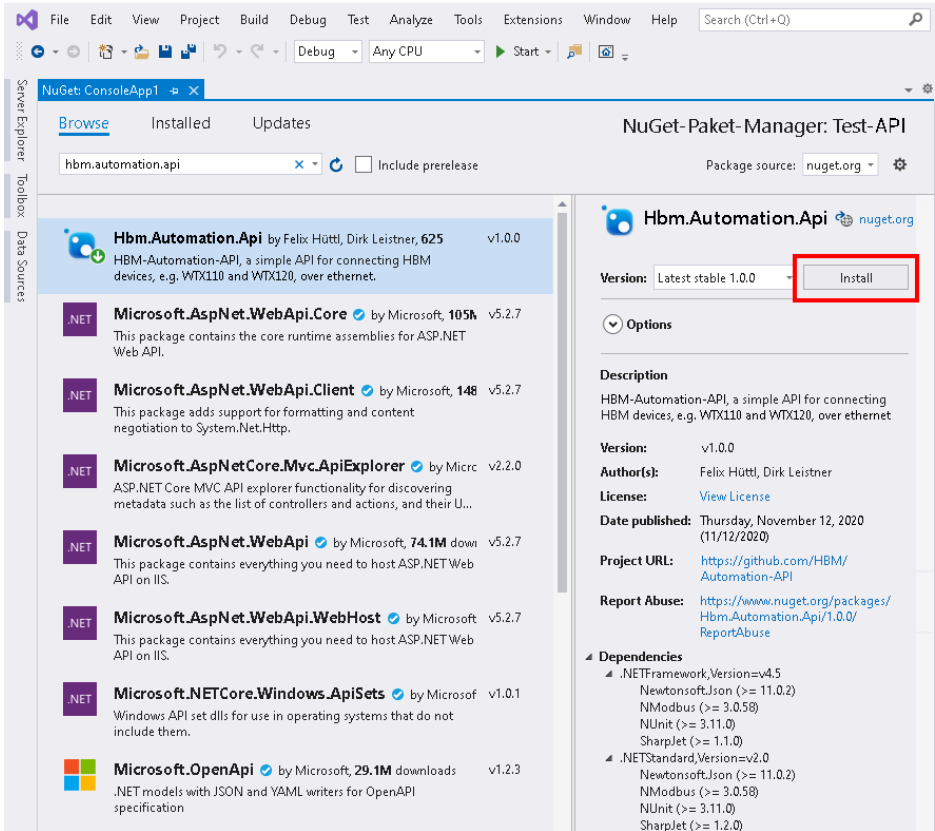


Fig. 4.3 Installing the HBM Automation API

5 Operating principle and functions of the HBM Automation API

This chapter provides an overview of the UML code structure of the HBM Automation API (*section 5.1*), as well as an explanation of the basic methods and properties you can use to create an application (*section 5.2*). For step-by-step instructions on how to create a working program, see *section 7.2, page 24*.

5.1 General structure of the HBM Automation API

Both the Modbus/TCP and Jet bus connections use the `INetConnection` as their interface, and use an Event Handler to indicate that new data has been read. `ProcessData` defines the properties (attributes) of values that can be displayed in the GUI application. `WTXModbus`, `WTXJet` and `DSEJet` are classes derived from `BaseWTDevice`. `WTXModbus`, for example, contains the instances of `DataLimitSwitch`, `DataDigitalIO` or `DataFiller` (for an application in WTX fill mode). `DataFillerExtended` is a special instance of `WTXJet`, because additional data is available via Jet.

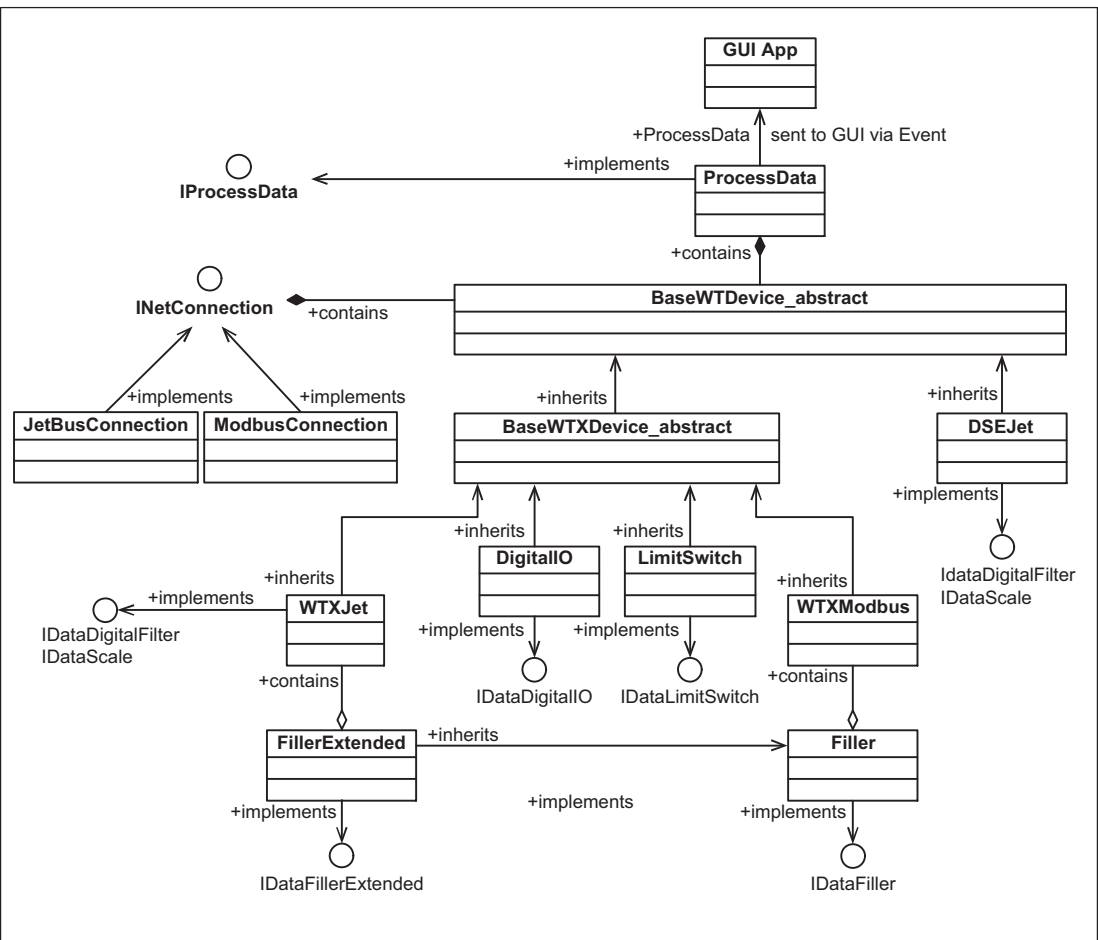


Fig. 5.1 UML diagram of the HBM Automation API

5.2 Basic functions in BaseWTDevice

The following table briefly sets out the key properties and methods for BaseWTDevice. You will find a complete list, with command syntax for the different programming languages, in the HBM Automation API documentation, which can be downloaded from GitHub:

<https://github.com/HBM/Automation-API> in "Help" folder.

Parameter	Name	Description
Weight value		
Property	PrintableWeight	Current weight values (net, gross and tare) as text, e.g. "12.3"
Property	Weight	Current weight values (net, gross and tare) as number (double)
Property	Unit	Sets or reads the unit, e.g. "g", "kg", "t"
Weighing status		
Property	ScaleRange	Reads the weighing range (1 ... 3)
Property	TareMode	Reads the signal type (gross or net)
Property	WeightStable	Reads whether there is an exact zero
Property	GeneralScaleError	Reads the general scale error, e.g. no sensor connected
Weighing		
Method	RecordWeight	Writes the current weight value to the Legal-for-Trade memory
Device information		
Property	Identification	Sets or reads the device ID, e.g. "WTX120"
Property	FirmwareVersion	Reads the firmware version
Property	SerialNumber	Reads the serial number

Parameter	Name	Description
Tare/Zero		
Property	ManualTareValue	Sets or reads the manual tare value
Method	SetGross	Sets the signal type to gross
Method	Tare	Tares the device (by measurement)
Method	TareManually	Tares the device manually
Method	Zero	Sets the device to zero (with measurement)
Setting		
Property	MaximumCapacity	Sets or reads the maximum weight
Property	CalibrationWeight	Sets or reads the calibration weight for the next calibration
Property	ZeroSignal	Sets or reads the zero value
Property	NominalSignal	Sets or reads the nominal value
Method	AdjustNominalSignal	Measure nominal signal
Method	AdjustNominalSignal WithCalibrationWeight ¹⁾	Measure and calculate nominal signal with specified weight
Method	AdjustZeroSignal	Measure zero value
Method	CalculateAdjustment	Set scaling with zero value and span
Cyclic process data		
Property	ProcessDataInterval	Sets or reads the interval for process updating
Property	ProcessData	Reads the current process data, e.g. measured value and status
Event	ProcessDataReceived	Indicates that new data is available, i.e. that the ProcessDataInterval has elapsed
Method	Restart	Restart for data update
Method	Stop	Stop data update

¹⁾ AdjustNominalSignalWithCalibrationWeights one word. The two-line formatting is dictated by the amount of space in the column.

Parameter	Name	Description
Connection		
Method	Connect	Depending on the transmitted parameters, a synchronous or asynchronous call to connect
Method	Disconnect	Depending on the transmitted parameters, a synchronous or asynchronous call to disconnect
Property	Connection	Reads the current connection to the device
Property	ConnectionType	Reads the connection type, e.g. JET or Modbus
Property	IsConnected	Reads whether the device is connected or not
Application mode		
Property	ApplicationMode	Sets or reads the application mode, e.g. default or fill mode

6 Examples

Make sure that the following conditions are met before trying out the sample applications:

- Your device must be connected to your PC via the same network segment.
- The device must be operational (e.g. sensor and supply voltage connected).
- You must have downloaded the sample data to Visual Studio as described in *section 4.1*.

The HBM Automation API (Hbm.Automation.Api) is included in the project samples, and does not need to be downloaded separately.

6.1 Example CommandLine

In the example, you establish a connection to the WTX120 weighing terminal via Modbus/TCP or Jet. You can only connect to the WTX110 weighing terminal and the DSE via Jet.

The example displays values including the net and gross weight in a console window. The display of new values is event-based, meaning new values are displayed as soon as the value changes. You can also trigger actions by clicking various buttons.

Running the sample application

- ▶ Right-click on **CommandLine** in the **Examples** folder in the **Solution Explorer** and select **Properties**, or double-click **Properties** (*Fig. 6.1*) directly.
- ▶ From the window on the left choose **Debug** and enter the **Command line arguments: Modbus** and the IP address for a connection via Modbus, **Jet** and the IP address for a connection via Jet (*Fig. 6.2*).
- ▶ From the dropdown box at the top of the window choose **CommandLine** (*Fig. 6.2*).
- ▶ To start the program, click the **Start** button to the right of the dropdown box.

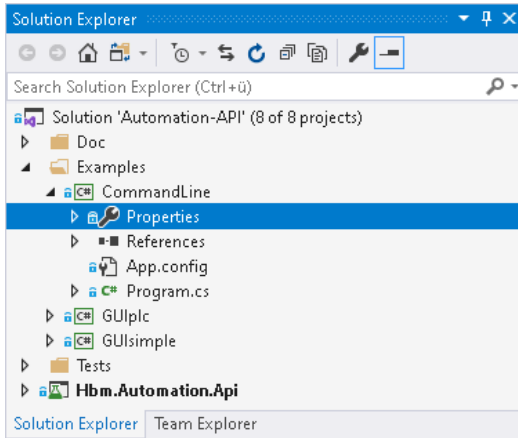


Fig. 6.1 *Selecting properties*

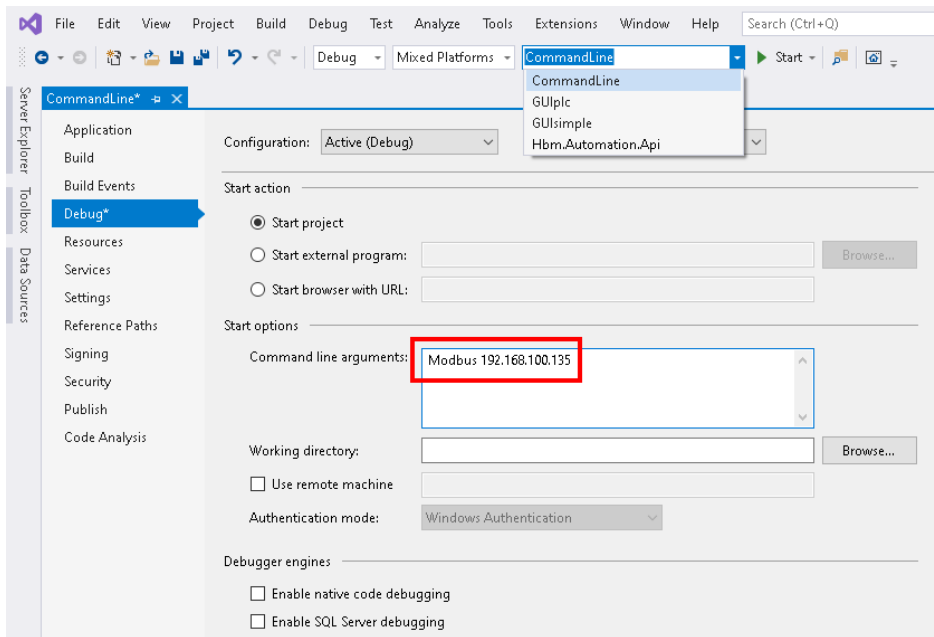


Fig. 6.2 *Entering the start parameters*

As soon as the program has been successfully created and run, a window opens showing the current device data.

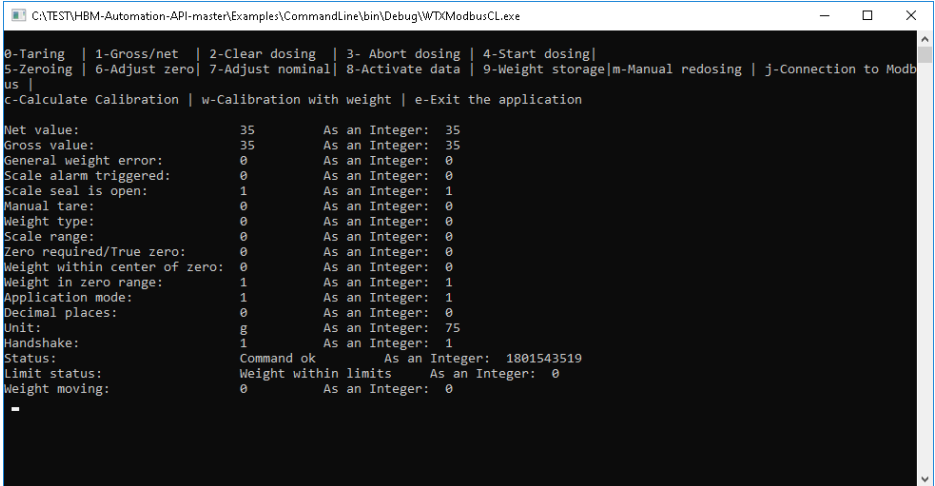


Fig. 6.3 CommandLine program output window

6.2 Example GUIsimple

In the example, you establish a connection to the WTX120 weighing terminal via Modbus/TCP or Jet. You can only connect to the WTX110 weighing terminal and the DSE via Jet.

The sample application generates a window in which gross, net and tare values are displayed and you can trigger zeroing or taring.

- ▶ From the dropdown box at the top of the window choose **GUIsimple**.
- ▶ To start the program, click the **Start** button to the right of the dropdown box.

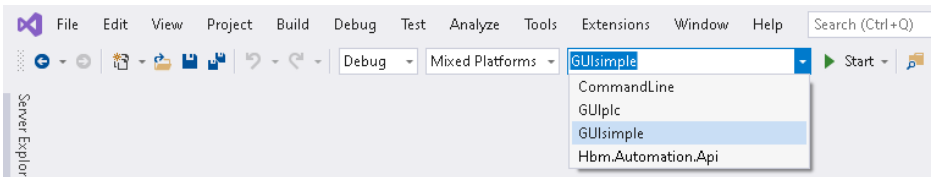


Fig. 6.4 Starting the GUIsimple sample application

As soon as the program has been successfully created and run, a window opens.

- ▶ Enter the IP address of the weighing terminal and the connection type (Fig. 6.5).

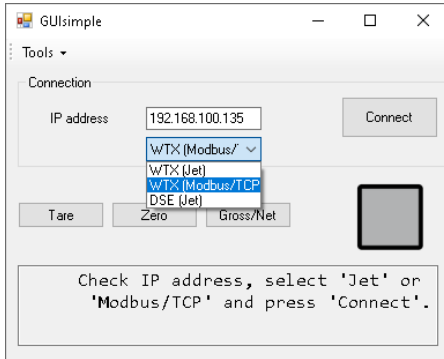


Fig. 6.5 Entering the start parameters

- ▶ Click on **Connect**.

When the connection has been established after a few seconds, the window changes and displays the values (Fig. 6.6).

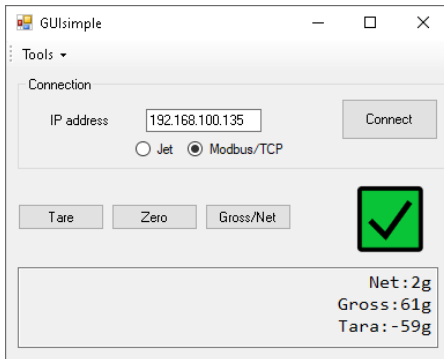


Fig. 6.6 Entering the start parameters

- ▶ Click on one of the buttons to run the relevant function.

6.3 Example GUIplc

In the example, you establish a connection to the WTX120 weighing terminal via Modbus/TCP.

The sample application generates a window in which all parameters and values are displayed and you can trigger various actions.

- From the dropdown box at the top of the window choose **GUIplc**.

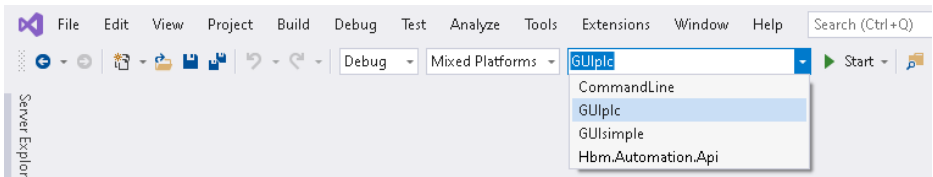


Fig. 6.7 Starting the GUIplc sample application

- Open the **GUIplc Properties** and in **Settings** specify the IP address to use (Fig. 6.8).

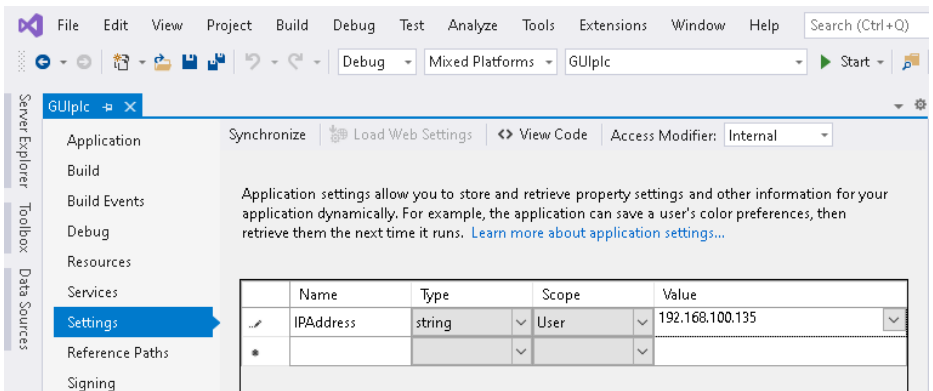


Fig. 6.8 Specifying the IP address

- To start the program, click the **Start** button at the top right.

As soon as the program has been successfully created and run, the program window opens (Fig. 6.9).

Input Word	Input Name	Input Type	Input Bit	Input Interface call routine	Input List call routine	Input Description	Input Value	Output Word	Output Bit
0	Measured Value	Int32	32Bit	IDevice_Values.Net Value	data.Sr[1]	Net measured	0	0	Control w
2	Measured Value	Int32	32Bit	IDevice_Values.Gross Value	data.Sr[2]	Gross measured	0	0	Control w
4	DS461-Weight st...	Bit	0	IDevice_Values.general_weight_error	data.Sr[3]	General weight er...	0	0	Control w
4	DS461-Weight st...	Bit	1	IDevice_Values.scale_alarm_triggered	data.Sr[4]	Scale alarm(s) trng...	0	0	Control w
4	DS461-Weight st...	Bits	2-3	IDevice_Values.limit_status	data.Sr[5]	Limit status	0	0	Control w
4	DS461-Weight st...	Bit	4	IDevice_Values.weight_moving	data.Sr[6]	Weight moving	0	0	Control w
4	DS461-Weight st...	Bit	5	IDevice_Values.scale_seal_is_open	data.Sr[7]	Scale seal is open	0	0	Control w
4	DS461-Weight st...	Bit	6	IDevice_Values.manual_tare	data.Sr[8]	Manual tare	0	0	Control w
4	DS461-Weight st...	Bit	7	IDevice_Values.weight_type	data.Sr[9]	Weight type	0	2	Manual ti
4	DS461-Weight st...	Bits	8-9	IDevice_Values.scale_range	data.Sr[10]	Scale range	0	4	Limit valu
4	DS461-Weight st...	Bit	10	IDevice_Values.zero_required	data.Sr[11]	Zero required	0	5	Limit valu
4	DS461-Weight st...	Bit	11	IDevice_Values.weight_within_the_center_of_zero	data.Sr[12]	Weight within the...	0	6	Limit valu
4	DS461-Weight st...	Bit	12	IDevice_Values.weight_in_zero_range	data.Sr[13]	Weight in zero ra...	0	8	Limit valu
5	Measured value ...	Bits	0-1	IDevice_Values.application_mode	data.Sr[14]	Application mode	0	10	Limit valu
5	Measured value ...	Bits	4-6	IDevice_Values.decimals	data.Sr[15]	Decimals	0	11	Limit valu
5	Measured value ...	Bits	7-8	IDevice_Values.unit	data.Sr[16]	Unit	0	12	Limit valu
5	Measured value ...	Bits	14	IDevice_Values.handshake	data.Sr[17]	Handshake	0	14	Limit valu
5	Measured value ...	Bit	15	IDevice_Values.status	data.Sr[18]	Status	0	16	Limit valu

Disconnected IP address Mode: TCP/Modbus 38 Jet/Modbus

Fig. 6.9 GUIplc program output window

7 Developing custom applications

The HBM Automation API gives you the ability to develop your own custom applications for your devices to optimally meet your requirements. To do so, you can either modify the sample applications detailed in *chapter 6* or develop your own completely new application.

This chapter describes the communication flow between the different classes and interfaces (*section 7.1*) as well as giving step-by-step instructions for creating your own custom application featuring the most important functions (*section 7.2*).

7.1 Communicating via the HBM Automation API

The following steps are needed to make a synchronous connection:

1. Create objects (instances) of the `ModbusTCPConnection` or `JetBusConnection` classes. When instantiating, you need to specify the IP address of the weighing terminal.

The class declarations include defaults for these values if you do not enter them. Modify the entries there as necessary.

2. Create objects of the classes `WTXModbus` or `WTXJet` (or `DSEJet`). When instantiating, you need to specify the connection object.
3. Call the `Connect(Timeout)` function of the `WTXModbus`, `WTXJet` or `DSEJet` objects after creating them.

```
ModbusTCPConnection _modConn = new ModbusTCPConnection(ipAddress);  
WTXModbus device = new WTXModbus(_modConn);  
device.Connect(2000);  
double netWeight = device.Weight.Net;
```

Fig. 7.1 Sample code for a synchronous connection via Modbus/TCP

For an asynchronous connection, such as to display the net weight from the example above continuously, you must additionally:

4. Use a Callback function. You specify the Callback method as an additional parameter when instantiating the WTXModbus, WTXJet or DSEJet object.

```
ModbusTCPConnection _modConn = new ModbusTCPConnection(ipAddress);
WTXModbus device = new WTXModbus(_modConn);
device.Connect();
double netWeight = device.Weight.Net;

private void updateCallback(object sender, ProcessDataReceivedEventArgs e)
{
    Dispatcher.BeginInvoke(new Action(delegate
    {
        double netWeight = e.ProcessData.Weight.Net;
    }));
}
```

Fig. 7.2 Sample code for an asynchronous connection

7.2 Creating your own custom application step-by-step

The following section will help you to develop your own custom application. To do so, you will create a simple GUI application that works similarly to the GUIsimple sample application detailed in *section 6.2*.

The sample application uses a Jet bus connection, but a connection via Modbus/TCP is programmed similarly.

7.2.1 Creating a new project

Start Visual Studio and open the HBM sample project as described in *section 4.1*. Then add another project to it (*Fig. 7.3*). Alternatively, you can create a completely new project and add the HBM Automation API as described in *section 4.2*. If you do that, use **Windows Forms App (.NET Framework)** as the project template (*Fig. 7.4*) and also specify a new storage location.

Once you have created the project, the Designer window opens with a form on which you can create the GUI.

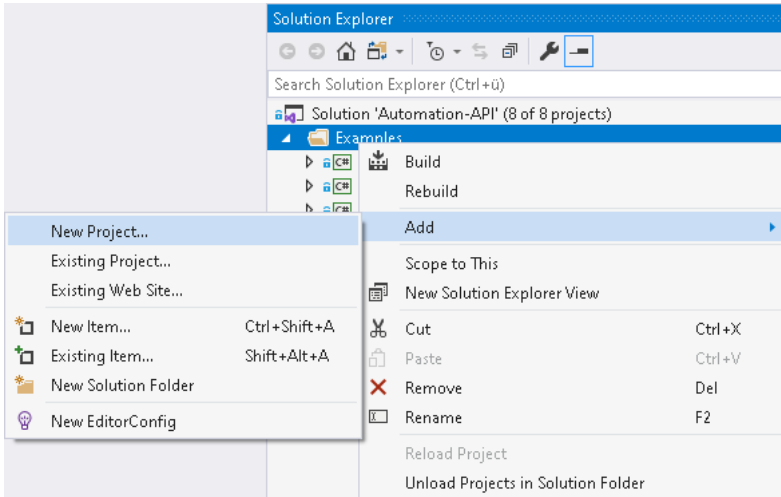


Fig. 7.3 Adding a new project

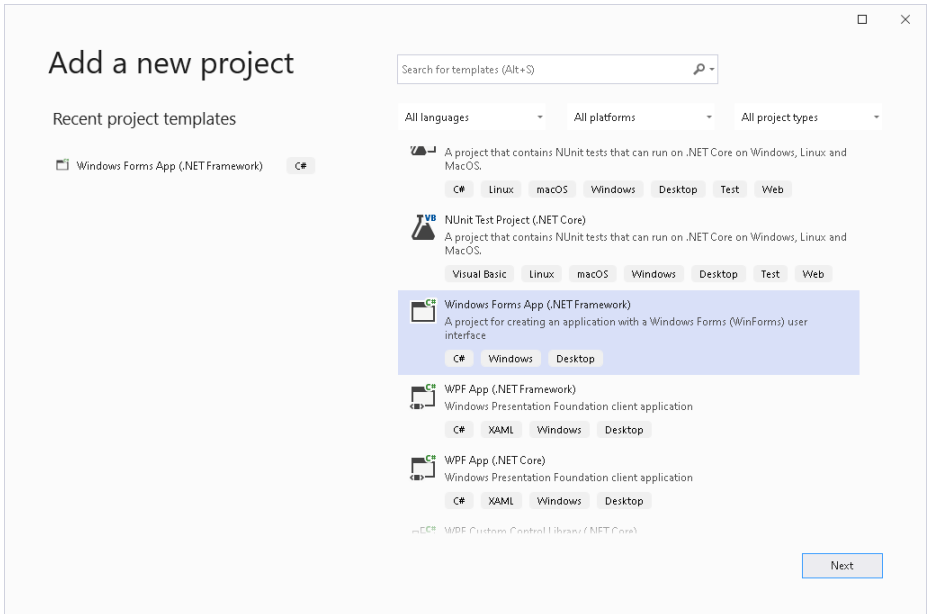


Fig. 7.4 Creating a Windows Forms app

7.2.2 Designing the GUI

Use the **Toolbox** (shown on the left, or open by pressing Ctrl+Alt+X) to design the GUI: The example in Fig. 7.5 includes **Button**, **Label**, **PictureBox**, **RadioButton** and **TextBox**, some of them more than once. Some objects also have modified properties, such as: **Font Size**, **Bold** type, or **MiddleCenter**/**MiddleLeft** alignment.

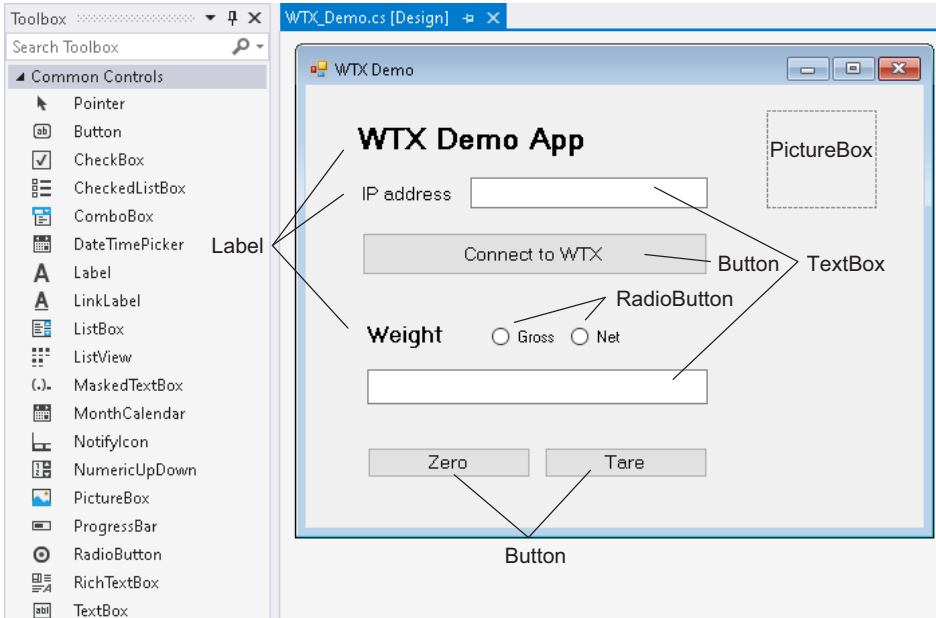


Fig. 7.5 The GUI (example)

Use the object **Properties** (Fig. 7.6) to specify the label (**Text**) and the **Name** for the objects listed in the table below so that the program can access them. You can of course also use different texts. If you want to use different names, you need to modify the code in the sample application accordingly. In the following examples the form has additionally been renamed "WTX_Demo" (in **Solution Explorer**).

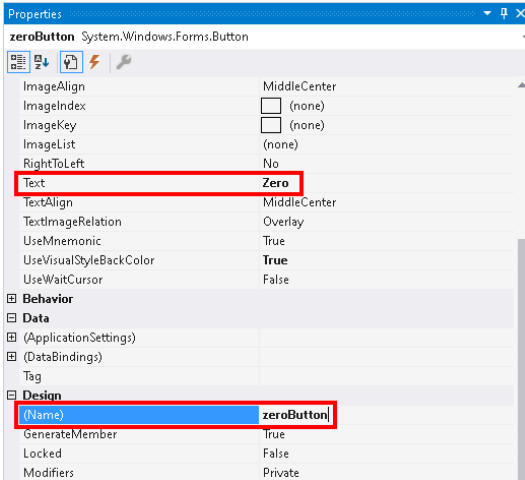


Fig. 7.6 Setting the "Text" and "Name" properties of the "Zero" button



Important

Do not confuse "Text" and "Name": "Text" is the label visible in the GUI; "Name" is used by the program for accessing.

Object	Label (text)	Name
Button	Connect to WTX	connectButton
	Tare	tareButton
	Zero	zeroButton
RadioButton	Net	netRadioButton
	Gross	grossRadioButton
TextBox	—	ipTextBox
	—	weightTextBox

7.2.3 Writing program code

Open the program code window by double-clicking on a free area within your GUI form. Part of the code – the basic code structure – has already been written.

- ▶ Add the components required for the HBM Automation API (Fig. 7.7).

Only Jet is used in the following, so there is actually no need to specify WTX.Modbus. It is only included here for the sake of completeness. The same applies conversely for WTX.Jet if you only want to use a Modbus connection. For the DSE, use DSE.Jet.

- ▶ Change the default names as you need (in **Solution Explorer** or via **Properties**). In the example WTX_Demo is used everywhere, so that is also the name for the namespace.

```
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.Automation.Api.Data;
using Hbm.Automation.Api.Weighing.WTX;
using Hbm.Automation.Api.Weighing.WTX.Jet;
using Hbm.Automation.Api.Weighing.WTX.Modbus;
namespace WTX_Demo
{
    public partial class WTX_Demo : Form
    {
        public WTX_Demo()
        {
            InitializeComponent();
        }
    }
}
```

Fig. 7.7 Start of code for WTX_Demo sample application

- ▶ If a Form1_Load or WTX_Demo_Load function exists, you can delete it; it is not needed here.
- ▶ Add the reference to the HBM Automation API (Fig. 7.8 and Fig. 7.9).

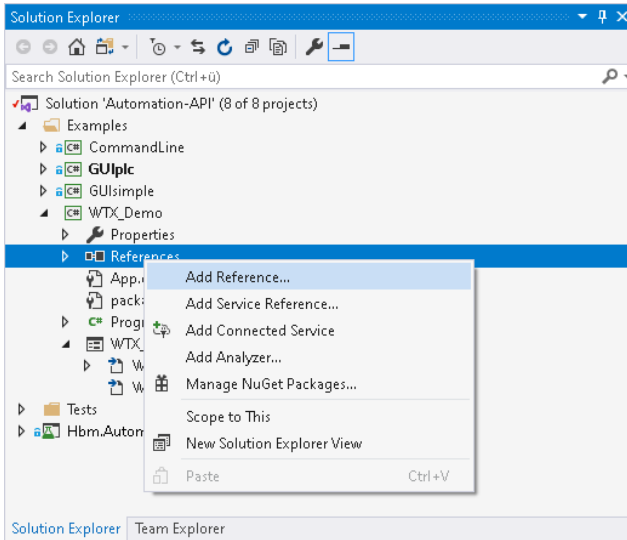


Fig. 7.8 Adding a reference

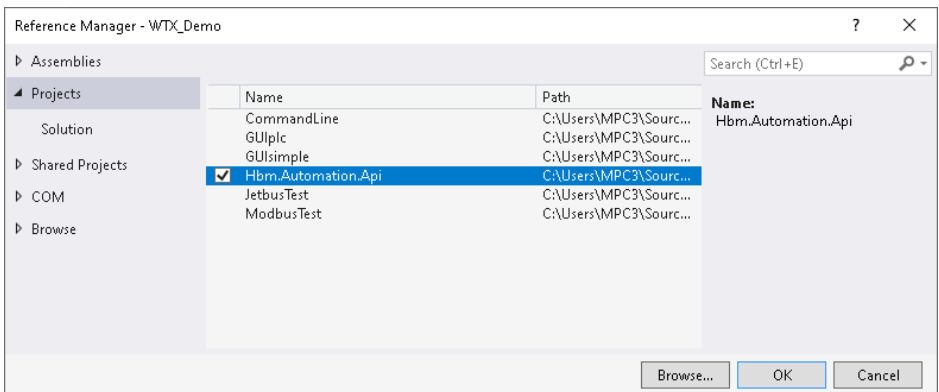


Fig. 7.9 Selecting the HBM Automation API

The "Connect to WTX" button connects the WTX weighing terminal to the PC. This requires the correct address to be entered in the box next to "IP address", otherwise the default address will be used.

- ▶ Double-click on the "Connect to WTX" button in the Designer window. This creates the connectButton_Click procedure (Fig. 7.10).

```
namespace WTX_Demo
{
    public partial class WTX_Demo : Form
    {
        public WTX_Demo()
        {
            InitializeComponent();
        }

        private void connectButton_Click(object sender, EventArgs e)
        {
        }
    }
}
```

Fig. 7.10 Procedure for the "Connect to WTX" button after creating it



Information

If you delete a code section that you created by double-clicking an object in the Design window, you will get an error message when you open the Design window again. As the code still exists in the Designer window (Resources.Designer), you must delete the code there as well. The easiest way to do this is to click on "Go to code" in the top right corner of the error message box.

- ▶ Declare the JetBusConnection and WTXDevice objects for Jet (WTXJet) in the header of the class (Fig. 7.11).
- ▶ As the IP address is needed for the connection, you also need a variable for it (Fig. 7.11).

```
namespace WTX_Demo
{
    public partial class WTX_Demo : Form
    {
        JetBusConnection _jetConnection;
        WTXJet _wtxDevice;
        string _ipAddress;
    }
}
```

Fig. 7.11 Declaring required objects and variables

In addition to the initialization, a default measured value type to display also needs to be set.

► Add the required code after initializing (Fig. 7.12).

So now you can write the code for the button.

► Read the IP address (Fig. 7.12).

► Specify it for the connection using the JetBusConnection object.

► This object serves as the input for WTXJet in the next step. The parameter 50 is the update rate in milliseconds.

```
namespace WTX_Demo
{
    public partial class WTX_Demo : Form
    {
        JetBusConnection _jetConnection;
        WTXJet _wtxDevice;
        string _ipAddress;

        public WTX_Demo()
        {
            InitializeComponent();
            netRadioButton.Checked = true;
        }

        private void connectButton_Click(object sender, EventArgs e)
        {
            _ipAddress = ipTextBox.Text;
            _jetConnection = new JetBusConnection(_ipAddress, "Administrator", "wtx");
            _wtxDevice = new WTXJet(_jetConnection, 50, update);

            try
            {
                _wtxDevice.Connect(2000);
                MessageBox.Show("WTX connected");
            }
            catch (Exception)
            {
                MessageBox.Show("Connection could not be established");
            }
        }
    }
}
```

Fig. 7.12 Declaring required objects and variables

► To make the connection you must use a try/catch construction, as various exceptions can occur when connecting. You must also specify a timeout – in the example 2000 (ms).

As you can see in the example, there are three parameters for WTXJet on the "_wtxDevice = ..." line. The third parameter is a function to provide continuous display of the weight. The Update function will continuously read the net or gross value from the WTX weighing terminal depending on the radio buttons. You can see the code for this function in *Fig. 7.13*.

```
private void update(object sender, ProcessDataReceivedEventArgs e)
{
    string weight;

    weightTextBox.BeginInvoke(new Action(() =>
    {
        if (this.netRadioButton.Checked)
            weight = e.ProcessData.PrintableWeight.Net;
        else
            weight = e.ProcessData.PrintableWeight.Gross;
        weightTextBox.Text = weight + " " + _wtxDevice.Unit;
    }));

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

Fig. 7.13 Update function

► Now write the code for the **Zero** and **Tare** buttons.

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

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

Fig. 7.14 The functions for the "Tare" and "Zero" buttons

In the example, the HBM logo has also been placed in the **PictureBox** via **Properties** (Fig. 7.15). To do this, click on the icon with the three dots on the **Image** line and import an image of your choice as a **Local Resource**.

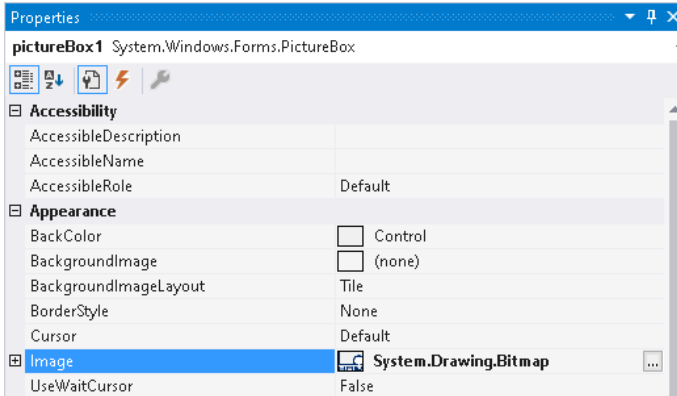


Fig. 7.15 Loading an image

The program is now finished. Select it at the top of the window and run it. Provided you haven't made any typing errors, the program will be created and started (Fig. 7.16).

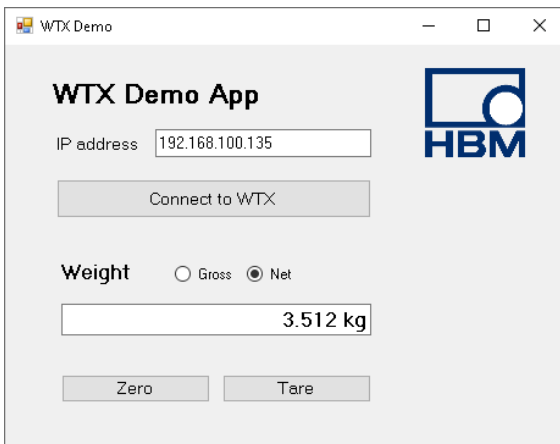


Fig. 7.16 The finished program

The complete program is set out again below (over two pages). Any lines that were inserted automatically and are not needed have been removed here so as to focus the program on the essential lines.

```

using System;
using System.Windows.Forms;
using Hbm.Automation.Api.Data;
using Hbm.Automation.Api.Weighing.WTX;
using Hbm.Automation.Api.Weighing.WTX.Jet;
namespace WTX_Demo
{
    public partial class WTX_Demo : Form
    {
        JetBusConnection _jetConnection;
        WTXJet _wtxDevice;
        string _ipAddress;

        public WTX_Demo()
        {
            InitializeComponent();
            netRadioButton.Checked = true;
        }

        private void connectButton_Click(object sender, EventArgs e)
        {
            _ipAddress = ipTextBox.Text;
            _jetConnection = new JetBusConnection(_ipAddress, "Administrator", "wtx");
            _wtxDevice = new WTXJet(_jetConnection, 50, update);
            try
            {
                _wtxDevice.Connect(2000);
                MessageBox.Show("WTX connected");
            }
            catch (Exception)
            {
                MessageBox.Show("Connection could not be established");
            }
        }

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

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

```

```

private void update(object sender, ProcessDataReceivedEventArgs e)
{
    string weight;

    weightTextBox.BeginInvoke(new Action(() =>
    {
        if (this.netRadioButton.Checked)
            weight = e.ProcessData.PrintableWeight.Net;
        else
            weight = e.ProcessData.PrintableWeight.Gross;
        weightTextBox.Text = weight + " " + _wtXDevice.Unit;
    }));

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

```

8 FAQs

What is open-source?

Open-source software is software whose source code has been made public and which can in most cases be used free of charge. It can usually also be modified, though sometimes subject to conditions. You are permitted to both modify the HBM Automation API and distribute it free of charge. No license fees or similar charges are payable. The only condition is that you include the copyright and authorization notices in your application (see GitHub).

What is an API?

An API (Application Programming Interface) is a programming interface to aid the creation of application programs. It is, in effect, a collection of functions that enable communication between different components of a system without having to know the specific instructions to be used. The API thus provides all the building blocks you need to communicate with the devices when developing an application program.

What is the difference between Modbus and Jet?

Modbus is a serial communication protocol, and is regarded as the standard for communication with a programmable logic controller (PLC). The Modbus protocol uses TCP port 502 by default.

The Jet bus (<https://jetbus.io>) is a programming framework developed for fast network communication, especially for the Internet of Things (IoT). It uses Publish/Subscribe messaging that allows multiple clients to connect to one server. Jet communication is founded on the TCP-based WebSocket protocol using TCP port 443.

What is WebSocket?

WebSocket is a communication protocol designed to run via TCP port 443. It provides bidirectional communication over a single TCP connection, enabling interactive communication between a web application (browser) and the WebSocket server. It also supports secure (encrypted) connections.

What is the difference between the HBM Automation API and standard industrial interfaces, such as for PLCs?

The HBM Automation API uses Jet to enable you to efficiently create your own custom solutions via a two-way encrypted TCP/IP connection. The programming blocks required for communication are built into the API.

PLC interfaces such as PROFINET[®], EtherNet/IP[™] or PROFIBUS[®] only provide a standard for fieldbus communication. So communication via those interfaces must be programmed first.

What does two-way encryption via https mean?

One-way encryption means that a system can only *encrypt*. An example is input of a password, which must be encrypted and then cross-checked against the authentication – no *decryption* is required. For the secure transmission of encrypted (measurement) data, however, decryption is also necessary to make the transmitted data readable again. This is known as two-way encryption, and requires that all network parties have both encryption and decryption functions to enable secure bidirectional communication. The encryption of the HBM Automation API is based on the HTTPS communication protocol, and uses the SSL cryptographic protocol. This authenticates the network parties using a handshake mechanism to protect against hacking.

9 Technical Support

If problems occur when working with the HBM Automation API, you can use the following services:

E-mail support

support@hbkworld.com

Telephone support

Telephone support is available on all working days from 09:00 to 5:00 PM (CET/CEST): +49 6151 803-0

The following options are also available: HBM Support and Sales International:
<https://www.hbm.com/en/0051/worldwide-contacts/>.

Headquarters worldwide

Europe:

Hottinger Brüel & Kjaer GmbH
Im Tiefen See 45, 64293 Darmstadt, Germany

North and South America

HBM, Inc., 19 Bartlett Street, Marlborough, MA 01752, USA
Tel. +1 800-578-4260 / +1 508-624-4500
Fax +1 508-485-7480
E-mail: info@usa.hbm.com

Asia

Hottinger Baldwin Measurement (Suzhou) Co., Ltd.
106 Heng Shan Road, Suzhou 215009, Jiangsu, PR China
Tel. +86 512-68247776, Fax +86 512-68259343
E-mail: hbmchina@hbm.com.cn

Operating Manual | **Bedienungsanleitung**

English

Deutsch

HBM Automation API

API for Industrial Electronics



1	Sicherheitshinweise	3
2	Verwendete Kennzeichnungen	4
3	Überblick	5
3.1	Systembeschreibung	5
3.2	Betriebsvoraussetzungen	6
3.2.1	Unterstützte Geräte	6
3.2.2	Benötigte Hardware	6
3.2.3	Benötigte Software	7
3.3	Verbindung zum PC	7
4	Installation	8
4.1	Installation der Daten für die Beispielprogramme	8
4.2	Installation der HBM Automation API für eigene Projekte	10
5	Arbeitsweise und Funktionen der HBM Automation API	12
5.1	Allgemeine Struktur der HBM Automation API	12
5.2	Basisfunktionen im BaseWTDevice	14
6	Beispiele	17
6.1	Beispiel CommandLine	17
6.2	Beispiel GUIsimple	19
6.3	Beispiel GUIplc	21
7	Eigene Anwendungen entwickeln	23
7.1	Kommunikation über die HBM Automation API aufbauen	23
7.2	Eine eigene Anwendung Schritt-für-Schritt erstellen	24
7.2.1	Neues Projekt erstellen	24
7.2.2	GUI-Fenster designen	26
7.2.3	Programm-Code schreiben	28
8	FAQ: Häufig gestellte Fragen	36
9	Technischer Support	38

1 Sicherheitshinweise


Bestimmungsgemäße Verwendung

Die HBM Automation API darf ausschließlich in Verbindung mit in dieser Anleitung genannten HBM-Geräten verwendet werden. Jeder darüber hinausgehende Gebrauch gilt als nicht bestimmungsgemäß.

Die HBM Automation API stellt die Funktionen der Geräte für die Software Visual Studio zur Verfügung. Die Bedienungsanleitungen der Geräte enthalten die zu beachtenden Sicherheitshinweise für den Betrieb der Geräte.

2 Verwendete Kennzeichnungen

Wichtige Hinweise sind besonders gekennzeichnet. Beachten Sie diese Hinweise unbedingt, um Unfälle und Sachschäden zu vermeiden.

Symbol	Bedeutung
 Information	Diese Kennzeichnung weist auf Informationen zum Produkt oder zur Handhabung des Produktes hin.
<i>Hervorhebung</i> <i>Siehe ...</i>	Kursive Schrift kennzeichnet Hervorhebungen im Text und kennzeichnet Verweise auf Kapitel, Bilder oder externe Dokumente und Dateien.
Gerät → Neu	Fette Schrift kennzeichnet Menüpunkte sowie Dialog- und Fenstertitel in Programmoberflächen. Pfeile zwischen Menüpunkten kennzeichnen die Reihenfolge, in der Menüs und Untermenüs aufgerufen werden
<i>Messrate</i>	Fett-kursive Schrift kennzeichnet Eingaben und Eingabefelder in Programmoberflächen.

3 Überblick

Stellen Sie sicher, dass Sie immer die für Ihr Gerät gültige Version der Bedienungsanleitung und die aktuelle Version der zugehörigen Firmware benutzen. Diese finden Sie in der aktuellsten Version stets auf der Website von HBM unter: <https://www.hbm.com/Downloads>.

Siehe auch *Kapitel 8, Seite 36*, in dem verschiedene der hier verwendeten Begriffe erklärt werden.

3.1 Systembeschreibung

Die HBM Automation API (Application Programming Interface) ist eine Open-Source-Schnittstelle, mit der Sie auf die Funktionalitäten verschiedener Geräte, z. B. der WTX-Wägeterminals, über Ethernet (TCP/IP) zugreifen können. Damit können Sie schnell eigene Anwendungsprogramme erstellen, die optimal an Ihre Anwendungen angepasst sind. Die API vereinfacht diese Erstellung dadurch, dass sie vordefinierte Klassen, Methoden und Variablen bereitstellt, die Sie direkt in die Entwicklungsumgebung Visual Studio einbinden können. Die Details, d. h., welche Befehle und Daten in welcher Kombination konkret an die Geräte geschickt werden müssen, sind daher nicht wichtig. Die API bietet Ihnen auch Möglichkeiten für die Protokollierung, Verarbeitung, gemeinsame Verwendung und Anzeige von Daten aus verschiedenen Quellen. Darüber hinaus enthält die API frei verfügbare Code-Beispiele als Vorlagen, die Sie für Ihre Anwendung und Anforderungen modifizieren können.

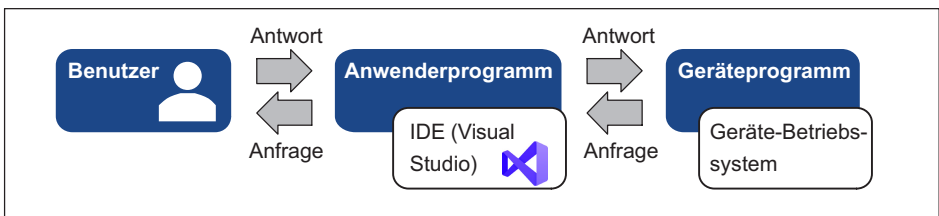


Abb. 3.1 Arbeitsweise der HBM Automation API

Sie können die Verbindung zu den Geräten über Ethernet TCP/IP mit dem JET-Bus oder bei einigen Geräten, z. B. bei WTX120, alternativ über die Modbus-Schnittstelle Ihres Steuerungssystems herstellen.

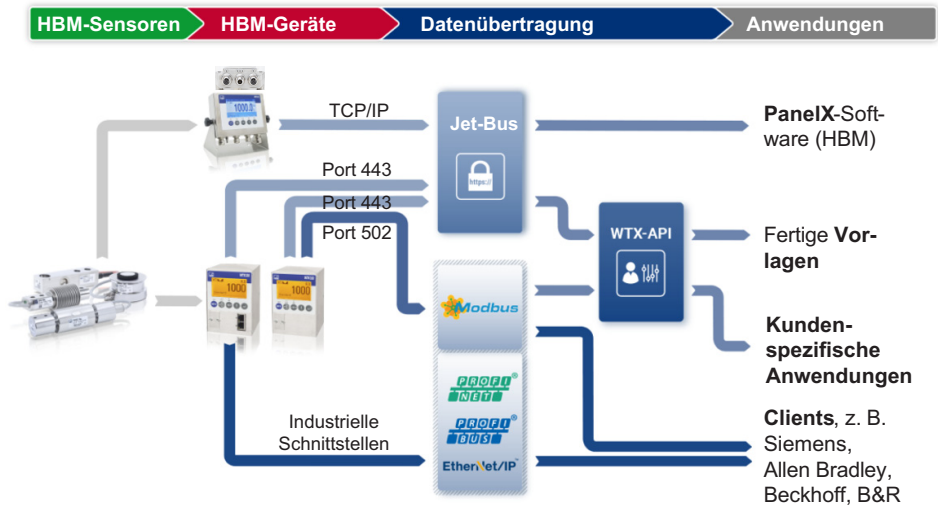


Abb. 3.2 Die HBM Automation API im Kontext der Messkette, Geräte und Aufnehmer sind als Beispiel eingezeichnet

3.2 Betriebsvoraussetzungen

Um die HBM Automation API verwenden zu können, benötigen Sie bestimmte Hard- und Software.

3.2.1 Unterstützte Geräte

- Wägeterminal WTX110
- Wägeterminal WTX120
- Digitale Sensorelektronik DSE-HIE

3.2.2 Benötigte Hardware

- Eines der unterstützten Geräte.
- Spannungsversorgung für das betreffende Gerät, z. B. 12 V ... 30 V_{DC} für die WTX-Geräte oder 15 V ... 30 V_{DC} für die DSE.

- An das HBM-Gerät angeschlossener Aufnehmer.
- PC mit LAN- oder WLAN-Anschluss zum gleichen Netzwerk wie das Gerät.
- Für Visual Studio sollte Ihr PC über ca. 50 GB freien Festplattenspeicher, 4 GB RAM-Speicher und eine 2 GHz-CPU mit mindestens 2 Kernen verfügen.

3.2.3 Benötigte Software

- Windows-Betriebssystem, mindestens Windows® 7 SP1.
- Visual Studio, mindestens Version 2013.
- .Net, mindestens Version 4.5.2.

3.3 Verbindung zum PC

Sie haben zwei Möglichkeiten, eine Verbindung zwischen HBM-Gerät und PC herzustellen:

1. Stellen Sie eine Verbindung über WLAN her.
2. Stellen Sie eine Ethernet-Verbindung über TCP/IP her. Entweder durch eine direkte Verbindung (peer-to-peer, P2P) oder über einen Switch.

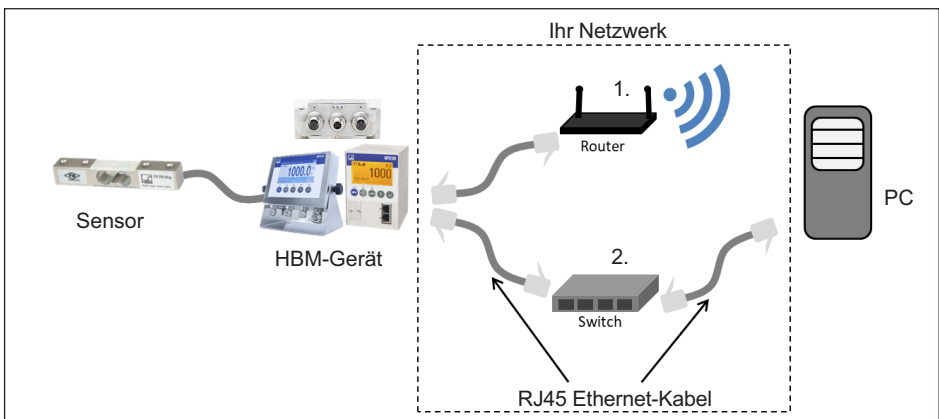


Abb. 3.3 Systemdarstellung mit den Verbindungsmöglichkeiten

4 Installation

Für alle Anwendungen benötigen Sie die (kostenlose) HBM Automation API (Hbm.Automation.Api), die Sie über den NuGet-Paketmanager in Visual Studio einbinden, siehe *Abschnitt 4.2*. Für einen leichteren Einstieg stellt Ihnen HBM drei lauffähige Beispielprogramme auf GitHub (ebenfalls kostenlos) zur Verfügung (siehe *Abschnitt 4.1*):

- Die Konsolen-Anwendung „CommandLine“.
- Die Windows-Forms-GUI-Anwendung „GUIplc“, die alle Ein- und Ausgaben für ein WTX-Wägeterminal enthält.
- Die Windows-Forms-GUI-Anwendung „GUIsimple“, die nur die wichtigsten Funktionen enthält.

Für die Beispiele müssen Sie die .NET-Desktopentwicklung in Visual Studio installieren (im Register **Workloads** des Visual Studio Installers).

4.1 Installation der Daten für die Beispielprogramme

Um alle Daten herunterladen zu können, müssen Sie zunächst einige Komponenten in Visual Studio installieren:

- ▶ Starten Sie den Visual Studio Installer.
- ▶ Klicken Sie auf **Ändern**.
- ▶ Gehen Sie zum Register **Einzelne Komponenten**.
- ▶ Aktivieren Sie im Abschnitt **Codetools** die Komponenten **Git für Windows** und **GitHub-Erweiterung für Visual Studio**, siehe *Abb. 4.1*.
- ▶ Falls nicht bereits installiert, aktivieren Sie auch den **NuGet-Paket-Manager** (*Abb. 4.1*).
- ▶ Klicken Sie zum Installieren auf **Ändern**.

Die Komponenten werden dann heruntergeladen und installiert.

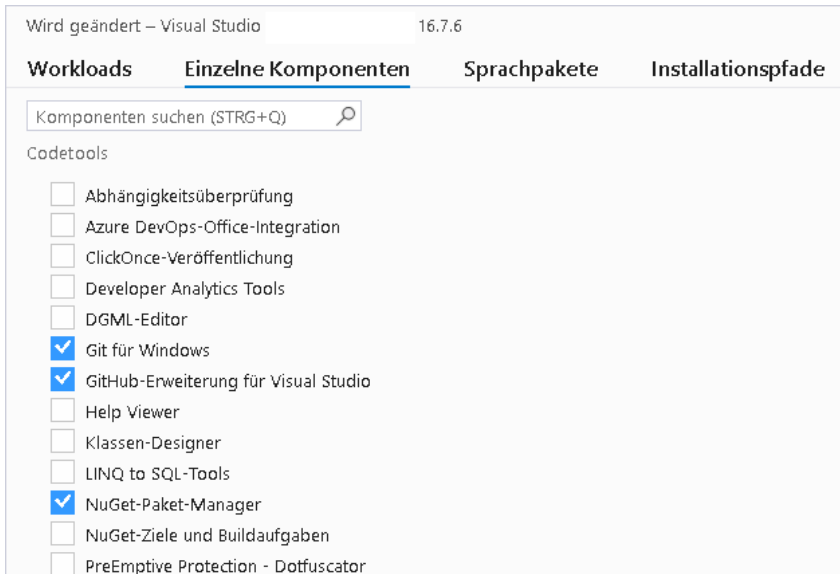


Abb. 4.1 Installation der benötigten Komponenten

Mit diesen Komponenten können Sie im nächsten Schritt die Beispieldateien von GitHub herunterladen.

- ▶ Starten Sie Visual Studio.
- ▶ Klicken Sie auf **Repository klonen**.
- ▶ Geben Sie **<https://github.com/HBM/Automation-API>** als Repositoryspeicherort ein, siehe Abb. 4.2.
- ▶ Legen Sie auch den lokalen Pfad fest, in dem das Projekt gespeichert werden soll (Unterverzeichnis „Source\Repos\Automation-API“ in Abb. 4.2).
- ▶ Klicken Sie auf **Klonen**.

Die Daten werden heruntergeladen und das Projekt geöffnet. Die Beschreibung der Beispiele finden Sie in *Kapitel 6*.

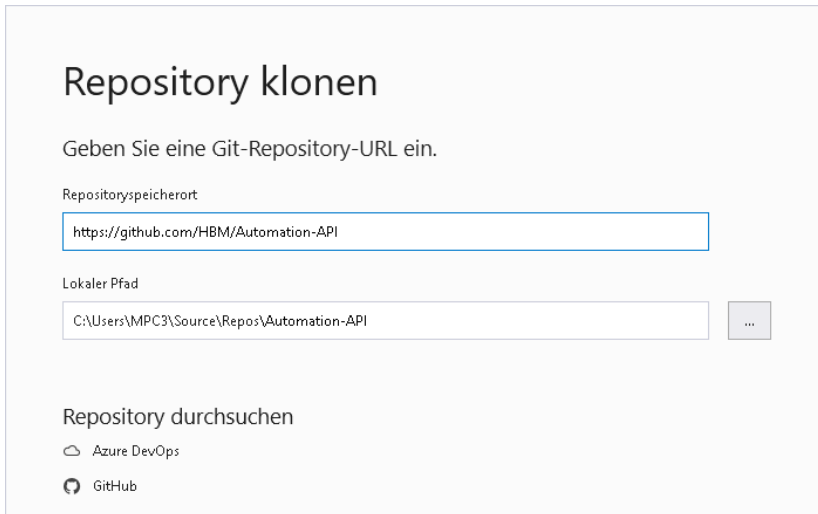


Abb. 4.2 Repository für die Beispielprogramme angeben

4.2 Installation der HBM Automation API für eigene Projekte

Falls Sie für Ihre eigenen Programme nur die API benötigen und der NuGet-Paket-Manager bereits installiert ist (siehe ansonsten *Abschnitt 4.1*), gehen Sie wie folgt vor:

- ▶ Starten Sie Visual Studio und erstellen Sie Ihr Projekt wie üblich.
- ▶ Rufen Sie mit **Projekt** → **NuGet-Pakete verwalten** den Paket-Manager auf.
- ▶ Suchen Sie nach **Hbm.Automation.Api**.
- ▶ Markieren Sie den Eintrag und klicken Sie auf **Installieren** (Abb. 4.3).

Die HBM Automation API wird heruntergeladen und steht in Ihrem Projekt zur Verfügung.

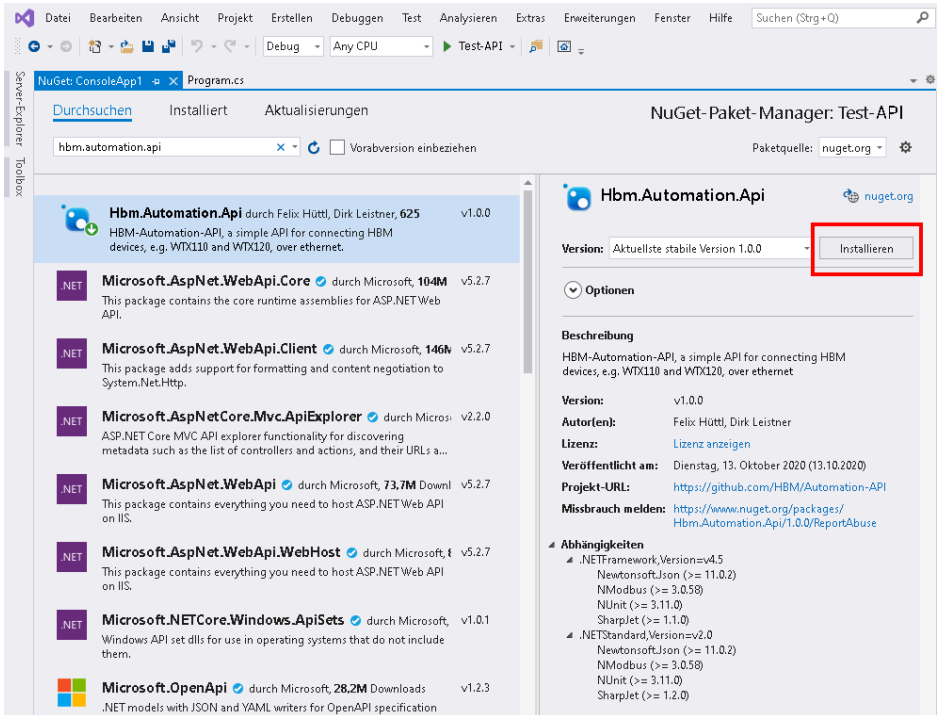


Abb. 4.3 Installation der HBM Automation API

5 Arbeitsweise und Funktionen der HBM Automation API

Dieses Kapitel enthält einen Überblick über die UML-Code-Struktur der HBM Automation API (*Abschnitt 5.1*) und eine Erläuterung der grundlegenden Methoden und Eigenschaften, die Sie für die Erstellung einer Anwendung verwenden können (*Abschnitt 5.2*). Eine Schritt-für-Schritt-Anleitung, wie Sie ein funktionierendes Programm erstellen, finden Sie in *Abschnitt 7.2, Seite 24*.

5.1 Allgemeine Struktur der HBM Automation API

Sowohl die Verbindung über Modbus/TCP als auch über Jet-Bus benutzt als Schnittstelle die `INetConnection` und verwendet einen Event-Handler, um anzuzeigen, dass neue Daten gelesen wurden. `ProcessData` definiert die Eigenschaften (Attribute) von Werten, die in der GUI-Anwendung angezeigt werden können. `WTXModbus`, `WTXJet` und `DSEJet` sind von `BaseWTDevice` abgeleitete Klassen. Dabei enthält `WTXModbus` z. B. die Instanzen von `DataLimitSwitch`, `DataDigitalIO` oder `DataFiller` (für eine Anwendung im Füllmodus der WTX). `DataFillerExtended` ist eine spezielle Instanz von `WTXJet`, da über den Jet-Bus zusätzliche Daten zur Verfügung stehen.

5.2 Basisfunktionen im BaseWTDevice

Die folgende Tabelle beschreibt kurz die wichtigsten Eigenschaften und Methoden für das BaseWTDevice. Eine vollständige Auflistung mit Befehlssyntax für die verschiedenen Programmiersprachen finden Sie in der Dokumentation zum HBM Automation API, die Sie über GitHub herunterladen können:

<https://github.com/HBM/Automation-API> im Verzeichnis „Help“.

Parameter	Name	Beschreibung
Gewichtswert		
Eigenschaft	PrintableWeight	Aktuelle Gewichtswerte (Netto, Brutto und Tara) als Text, z. B. "12,3"
Eigenschaft	Weight	Aktuelle Gewichtswerte (Netto, Brutto und Tara) als Zahl (double)
Eigenschaft	Unit	Setzt oder liest die Einheit, z. B. "g", "kg", "t"
Status		
Verwiegung		
Eigenschaft	ScaleRange	Liest den Wägebereich (1 ... 3)
Eigenschaft	TareMode	Liest den Signaltyp (Brutto oder Netto)
Eigenschaft	WeightStable	Liest, ob eine genaue Null vorliegt
Eigenschaft	GeneralScaleError	Liest den allgemeinen Waagenfehler, z. B. kein Sensor angeschlossen
Verwiegung		
Methode	RecordWeight	Schreibt den aktuellen Gewichtswert in den Legal-for-Trade-Speicher
Geräteinformation		
Eigenschaft	Identification	Setzt oder liest die Geräte-ID, z. B. "WTX120"
Eigenschaft	FirmwareVersion	Liest die Firmwareversion
Eigenschaft	SerialNumber	Liest die Seriennummer

Parameter	Name	Beschreibung
Tarieren/Nullstellen		
Eigenschaft	ManualTareValue	Setzt oder liest den manuellen Tara-wert
Methode	SetGross	Setzt den Signaltyp auf Brutto
Methode	Tare	Tariert das Gerät (durch Messung)
Methode	TareManually	Tariert das Gerät manuell
Methode	Zero	Stellt das Gerät auf null (mit Messung)
Einstellung		
Eigenschaft	MaximumCapacity	Setzt oder liest das Maximalgewicht
Eigenschaft	CalibrationWeight	Setzt oder liest das Kalibriergewicht für die nächste Kalibrierung
Eigenschaft	ZeroSignal	Setzt oder liest den Nullwert
Eigenschaft	NominalSignal	Setzt oder liest den Nennwert
Methode	AdjustNominalSignal	Nennsignal messen
Methode	AdjustNominalSignal WithCalibrationWeight ¹⁾	Nennsignal mit angegebenem Gewicht messen und berechnen
Methode	AdjustZeroSignal	Nullwert messen
Methode	CalculateAdjustment	Skalierung mit Nullwert und Spanne setzen
Zyklische Prozessdaten		
Eigenschaft	ProcessDataInterval	Setzt oder liest das Intervall für die Prozessaktualisierung
Eigenschaft	ProcessData	Liest die aktuellen Prozessdaten, z. B. Messwert und Status
Ereignis	ProcessDataReceived	Zeigt an, dass neue Daten vorliegen, d. h., dass das ProcessDataInterval abgelaufen ist
Methode	Restart	Neustart für die Datenaktualisierung
Methode	Stop	Datenaktualisierung stoppen

¹⁾ AdjustNominalSignalWithCalibrationWeight ist ein Wort, die Schreibweise in zwei Zeilen ist durch den Platz in der Spalte bedingt.

Parameter	Name	Beschreibung
Verbindung		
Methode	Connect	Abhängig von den übergebenen Parametern ein synchroner oder asynchroner Aufruf zum Verbinden
Methode	Disconnect	Abhängig von den übergebenen Parametern ein synchroner oder asynchroner Aufruf zum Trennen
Eigenschaft	Connection	Liest die aktuelle Verbindung zum Gerät
Eigenschaft	ConnectionType	Liest den Verbindungstyp, z. B. JET oder Modbus
Eigenschaft	IsConnected	Liest, ob das Gerät verbunden ist oder nicht
Anwendungsmodus		
Eigenschaft	ApplicationMode	Setzt oder liest den Anwendungsmodus, z. B. Standard oder Füllmodus

6 Beispiele

Stellen Sie sicher, dass folgende Bedingungen erfüllt sind, bevor Sie die Beispiele ausprobieren:

- Ihr Gerät muss mit Ihrem PC über das gleiche Netzwerksegment verbunden sein.
- Das Gerät muss funktionsfähig sein (z. B. Sensor und Speisespannung angeschlossen).
- Sie müssen die Beispieldaten wie in *Abschnitt 4.1* beschrieben in Visual Studio heruntergeladen haben.

Die HBM Automation API (Hbm.Automation.Api) ist in den Beispielen des Projekts enthalten und muss nicht separat heruntergeladen werden.

6.1 Beispiel CommandLine

Im Beispiel stellen Sie eine Verbindung über Modbus/TCP oder Jet-Bus zum Wägeterminal WTX120 her. Beim Wägeterminal WTX110 und bei der DSE ist nur die Verbindung über Jet-Bus möglich.

Das Beispiel stellt Werte wie Netto- und Bruttogewicht in einem Konsolenfenster dar. Die Anzeige neuer Werte ist ereignisbasiert, d. h., neue Werte werden angezeigt, sobald sich der Wert ändert. Über verschiedene Tasten können Sie auch Aktionen auslösen.

Ausführen des Beispiels

- ▶ Klicken Sie mit der rechten Maustaste auf **CommandLine** im Ordner **Examples** im **Projektmappen-Explorer** und wählen Sie **Eigenschaften** oder doppelklicken Sie direkt auf **Properties** (*Abb. 6.1*).
- ▶ Tragen Sie im linken Fenster bei **Debuggen** die **Befehlszeilenargumente** ein: **Modbus** und die IP-Adresse für eine Verbindung über Modbus, **Jet** und die IP-Adresse für eine Verbindung über Jet-Bus (*Abb. 6.2*).
- ▶ Wählen Sie oben im Fenster **CommandLine** aus (*Abb. 6.2*).
- ▶ Starten Sie das Programm durch einen Klick auf **Starten** rechts daneben.

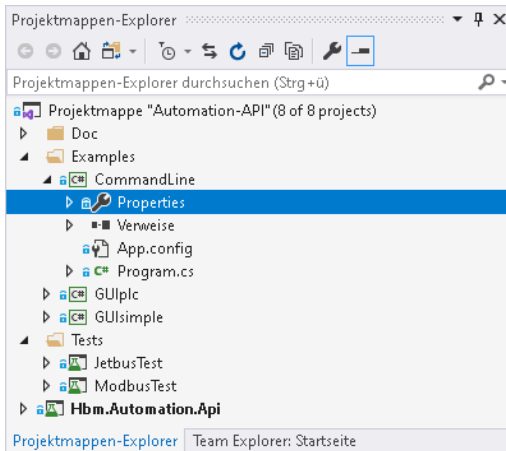


Abb. 6.1 Eigenschaften aufrufen

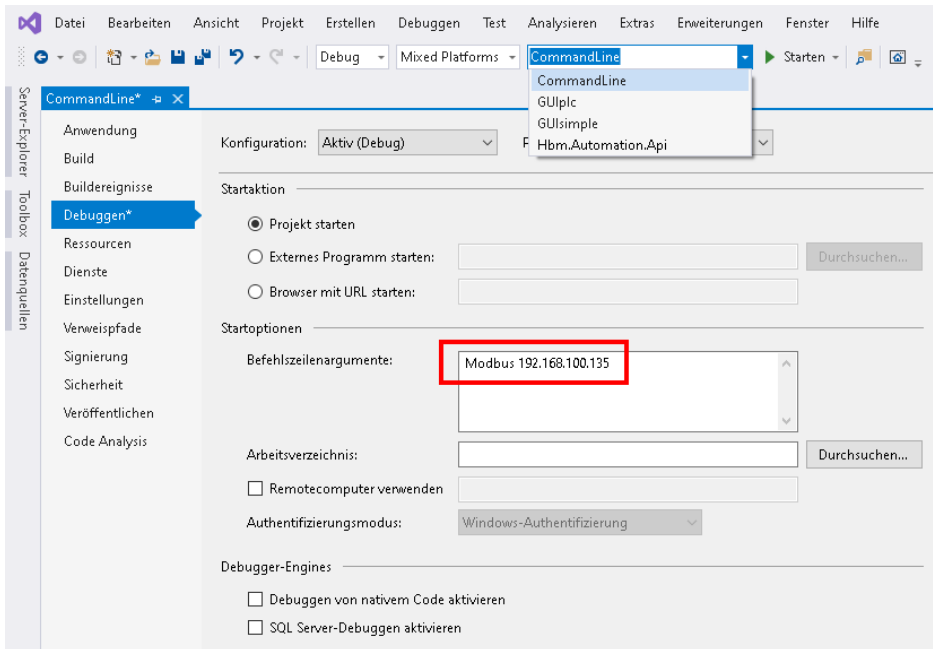


Abb. 6.2 Eingeben der Startparameter

Sobald das Programm erfolgreich erstellt wurde und ausgeführt wird, öffnet sich ein Fenster, das die aktuellen Werte des Gerätes zeigt.

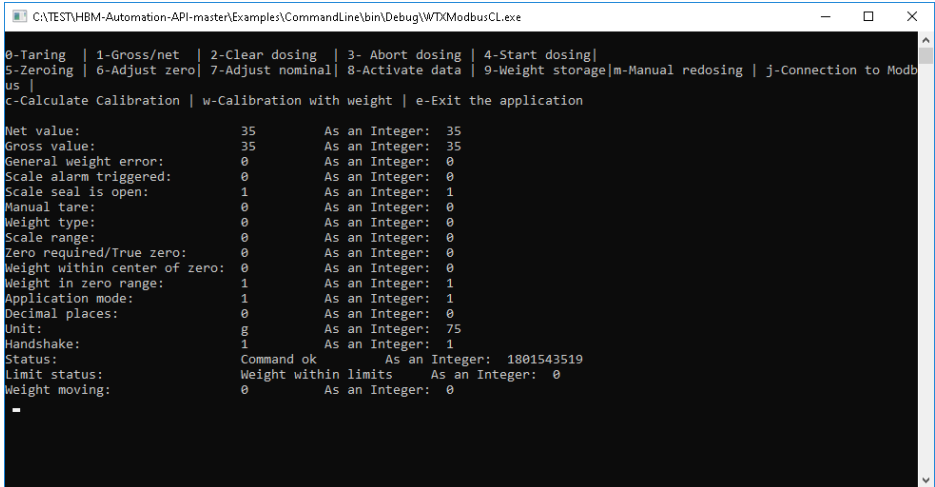


Abb. 6.3 Ausgabefenster des Programms CommandLine

6.2 Beispiel GUIsimple

Im Beispiel stellen Sie eine Verbindung über Modbus/TCP oder Jet-Bus zum Wägeterminal WTX120 her. Beim Wägeterminal WTX110 und der DSE ist nur die Verbindung über Jet-Bus möglich.

Das Beispiel erzeugt ein Fenster, in dem Brutto-, Netto- und Tarawert angezeigt werden und Sie einen Nullabgleich oder eine Tarierung auslösen können.

- ▶ Wählen Sie oben im Fenster **GUIsimple** aus.
- ▶ Starten Sie das Programm durch einen Klick auf **Starten** rechts daneben.

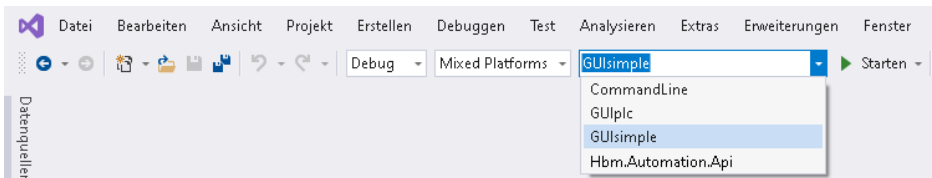


Abb. 6.4 Beispielprogramm GUIsimple starten

Sobald das Programm erfolgreich erstellt wurde und ausgeführt wird, öffnet sich ein Fenster.

- Geben Sie die IP-Adresse des Wägeterminals und den Verbindungstyp an (Abb. 6.5).

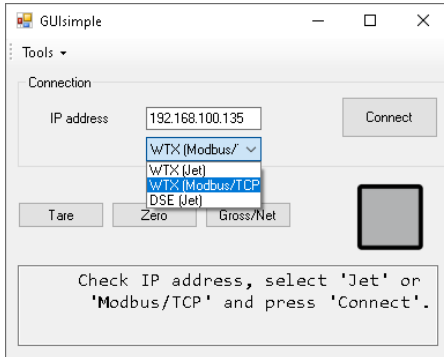


Abb. 6.5 Startparameter eingeben

- Klicken Sie auf **Connect**.

Wenn nach einigen Sekunden die Verbindung aufgebaut wurde, ändert sich das Fenster und zeigt die Werte an (Abb. 6.6).

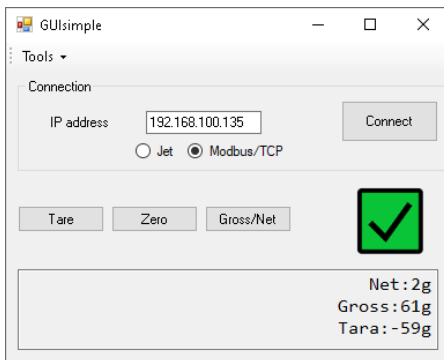


Abb. 6.6 Startparameter eingeben

- Klicken Sie auf eine der Schaltflächen, um die betreffende Funktion auszuführen.

6.3 Beispiel GUIplc

Im Beispiel stellen Sie eine Verbindung über Modbus/TCP zum Wägeterminal WTX120 her.

Das Beispiel erzeugt ein Fenster, in dem alle Parameter und Werte angezeigt werden und Sie verschiedene Aktionen auslösen können.

- ▶ Wählen Sie oben im Fenster **GUIplc** aus.

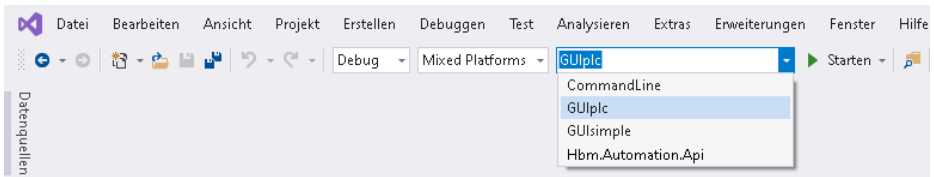


Abb. 6.7 Beispielprogramm GUIplc starten

- ▶ Öffnen Sie die Eigenschaften von GUIplc (**Properties**) und geben Sie bei **Einstellungen** die verwendete IP-Adresse an (Abb. 6.8).

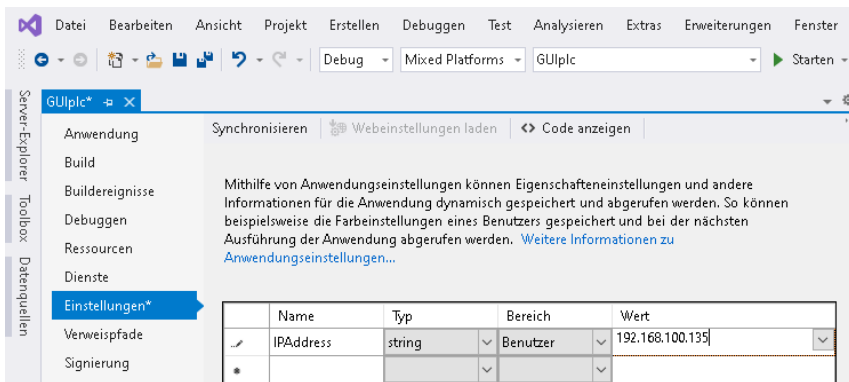


Abb. 6.8 IP-Adresse angeben

- ▶ Starten Sie das Programm durch einen Klick auf **Starten** oben rechts.

Sobald das Programm erfolgreich erstellt wurde und ausgeführt wird, öffnet sich das Programmfenster (Abb. 6.9).

Input Word	Input Name	Input Type	Input Bit	Input Interface call routine	Input List call routine	Input Description	Input Value	Output Word	Output Bit
0	Measured Value	Int32	32Bit	IDevice_Values.Net Value	data.Sr[1]	Net measured	0	0	Control w
2	Measured Value	Int32	32Bit	IDevice_Values.Gross Value	data.Sr[2]	Gross measured	0	0	Control w
4	DS461-Weight st...	Bit	0	IDevice_Values.general_weight_error	data.Sr[3]	General weight er...	0	0	Control w
4	DS461-Weight st...	Bit	1	IDevice_Values.scale_alarm_triggered	data.Sr[4]	Scale alarm(s) trng...	0	0	Control w
4	DS461-Weight st...	Bits	2-3	IDevice_Values.limit_status	data.Sr[5]	Limit status	0	0	Control w
4	DS461-Weight st...	Bit	4	IDevice_Values.weight_moving	data.Sr[6]	Weight moving	0	0	Control w
4	DS461-Weight st...	Bit	5	IDevice_Values.scale_seal_is_open	data.Sr[7]	Scale seal is open	0	0	Control w
4	DS461-Weight st...	Bit	6	IDevice_Values.manual_tare	data.Sr[8]	Manual tare	0	0	Control w
4	DS461-Weight st...	Bit	7	IDevice_Values.weight_type	data.Sr[9]	Weight type	0	2	Manual ti
4	DS461-Weight st...	Bits	8-9	IDevice_Values.scale_range	data.Sr[10]	Scale range	0	4	Limit valu
4	DS461-Weight st...	Bit	10	IDevice_Values.zero_required	data.Sr[11]	Zero required	0	5	Limit valu
4	DS461-Weight st...	Bit	11	IDevice_Values.weight_within_the_center_of_zero	data.Sr[12]	Weight within the...	0	6	Limit valu
4	DS461-Weight st...	Bit	12	IDevice_Values.weight_in_zero_range	data.Sr[13]	Weight in zero ra...	0	8	Limit valu
5	Measured value ...	Bits	0-1	IDevice_Values.application_mode	data.Sr[14]	Application mode	0	10	Limit valu
5	Measured value ...	Bits	4-6	IDevice_Values.decimals	data.Sr[15]	Decimals	0	11	Limit valu
5	Measured value ...	Bits	7-8	IDevice_Values.unit	data.Sr[16]	Unit	0	12	Limit valu
5	Measured value ...	Bits	14	IDevice_Values.handshake	data.Sr[17]	Handshake	0	14	Limit valu
5	Measured value ...	Bit	15	IDevice_Values.status	data.Sr[18]	Status	0	16	Limit valu

Disconnected IP address Mode: TCP/Modbus 38 Jet/Modbus

Abb. 6.9 Ausgabefenster des Programms GUIplc

7 Eigene Anwendungen entwickeln

Die HBM Automation API gibt Ihnen die Möglichkeit, eigene Anwendungen für Ihre Geräte zu entwickeln, die Ihre Anforderungen optimal erfüllen. Sie können dazu entweder die Beispiele in *Kapitel 6* modifizieren oder Ihre Anwendung komplett neu entwickeln.

Dieses Kapitel enthält eine Beschreibung des Kommunikationsflusses zwischen den verschiedenen Klassen und Schnittstellen (*Abschnitt 7.1*) und eine Schritt-für-Schritt-Anleitung zum Erstellen einer eigenen Anwendung mit den wichtigsten Funktionen (*Abschnitt 7.2*).

7.1 Kommunikation über die HBM Automation API aufbauen

Die folgenden Schritte sind nötig, um eine synchrone Verbindung zu erzeugen:

1. Erzeugen Sie Objekte (Instanzen) der Klassen `ModbusTCPConnection` oder `JetBusConnection`. Bei der Instanziierung müssen Sie die IP-Adresse des Wägeterminals angeben.

In der Klassendeklaration sind Voreinstellungen für diese Werte definiert, falls keine Angaben erfolgen. Ändern Sie die Angaben bei Bedarf dort.

2. Erzeugen Sie Objekte der Klassen `WTXModbus` bzw. `WTXJet` (oder `DSEJet`). Die Instanziierung benötigt die Angabe des betreffenden Verbindungsobjektes.
3. Rufen Sie die `Connect(Timeout)`-Funktion der `WTXModbus`- bzw. `WTXJet`- oder `DSEJet`-Objekte auf, nachdem diese erzeugt wurden.

```
ModbusTCPConnection _modConn = new ModbusTCPConnection(ipAddress);  
WTXModbus device = new WTXModbus(_modConn);  
device.Connect(2000);  
double netWeight = device.Weight.Net;
```

Abb. 7.1 Codebeispiel für eine synchrone Verbindung über Modbus/TCP

Für eine asynchrone Verbindung, z. B. um das Nettogewicht aus dem Beispiel oben kontinuierlich darstellen zu können, müssen Sie:

- Zusätzlich eine Callback-Funktion verwenden. Die Callback-Methode geben Sie als weiteren Parameter bei der Instanziierung des WTXModbus- bzw. WTXJet- oder DSEJet-Objektes an.

```
ModbusTCPConnection _modConn = new ModbusTCPConnection(ipAddress);
WTXModbus device = new WTXModbus(_modConn);
device.Connect();
double netWeight = device.Weight.Net;

private void updateCallback(object sender, ProcessDataReceivedEventArgs e)
{
    Dispatcher.BeginInvoke(new Action(delegate
    {
        double netWeight = e.ProcessData.Weight.Net;
    }));
}
```

Abb. 7.2 Codebeispiel für eine asynchrone Verbindung

7.2 Eine eigene Anwendung Schritt-für-Schritt erstellen

Der folgende Abschnitt unterstützt Sie bei der Entwicklung einer eigenen Anwendung. Dazu erstellen Sie nachfolgend eine einfache GUI-Anwendung, die ähnlich wie das Beispiel GUIsimple aus *Abschnitt 6.2* funktioniert.

Das Beispiel verwendet eine Jet-Bus-Verbindung. Eine Verbindung über Modbus/TCP wird jedoch ähnlich programmiert.

7.2.1 Neues Projekt erstellen

Starten Sie Visual Studio und öffnen Sie das Beispielprojekt von HBM wie in *Abschnitt 4.1* beschrieben. Fügen Sie diesem dann ein weiteres Projekt hinzu (*Abb. 7.3*). Sie können aber auch ein ganz neues Projekt anlegen und wie in *Abschnitt 4.2* beschrieben die HBM Automation API hinzufügen. Verwenden Sie in diesem Fall **Windows-Forms-App (.NET Framework)** als Projektvorlage (*Abb. 7.4*) und geben Sie auch einen neuen Speicherort an.

Nach dem Erzeugen des Projektes öffnet sich das Entwurfswindow mit einer Form, auf der Sie das GUI anlegen können.

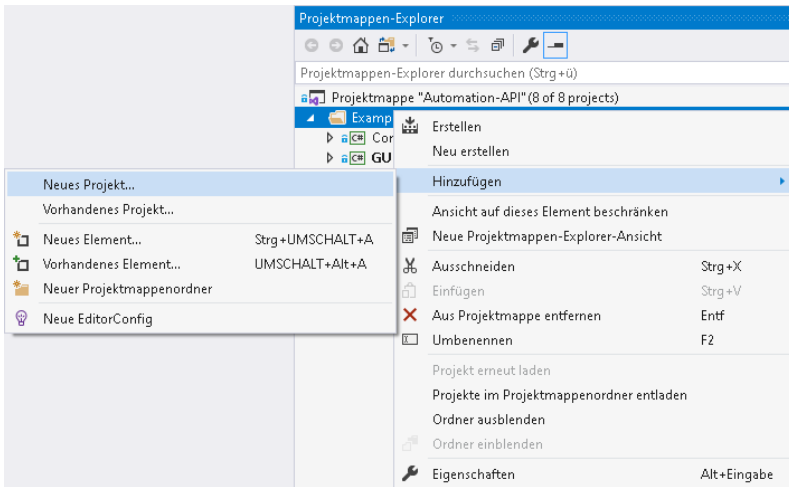


Abb. 7.3 Neues Projekt hinzufügen

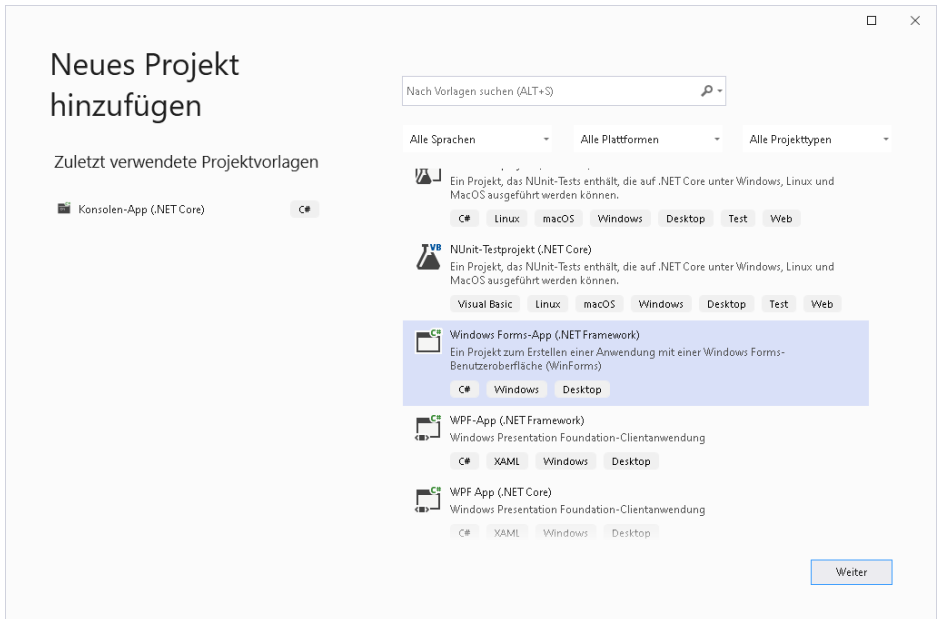


Abb. 7.4 Windows-Forms-App erstellen

7.2.2 GUI-Fenster designen

Erzeugen Sie mithilfe der **Toolbox** (linke Seite oder Strg+Alt+X) die Oberfläche: im Beispiel in *Abb. 7.5* sind (teils mehrfach) **Button**, **Label**, **PictureBox**, **RadioButton** und **TextBox** vorhanden. Einige Objekte haben auch veränderte Eigenschaften: **Font Size** (Schriftgröße) oder die Einstellung **Bold** (Fett) sowie die Ausrichtung **MiddleCenter** (Mitte-Zentriert) oder **MiddleLeft** (Mitte-Links).

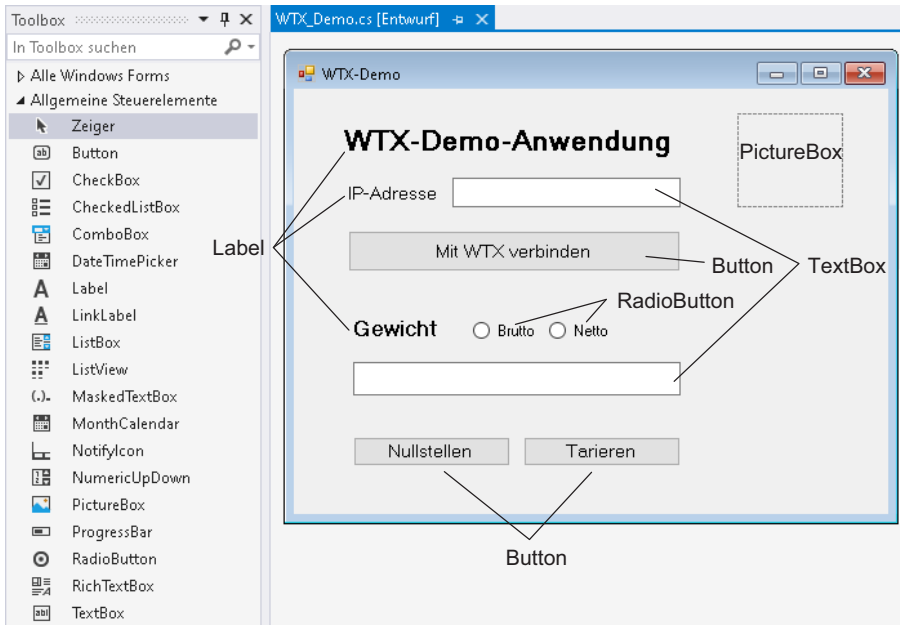


Abb. 7.5 Das GUI (Beispiel)

Geben Sie über das Fenster **Eigenschaften** der Objekte (*Abb. 7.6*) zusätzlich zur Beschriftung (**Text**) auch den **Namen** für die in der nachfolgenden Tabelle aufgeführten Objekte an, damit das Programm darauf zugreifen kann. Selbstverständlich können Sie auch andere Texte verwenden. Falls Sie andere Namen verwenden möchten, müssen Sie den Code im Beispiel entsprechend anpassen. Außerdem wurde in den folgenden Beispielen die Form in „WTX_Demo“ umbenannt (im **Projektmappen-Explorer**).

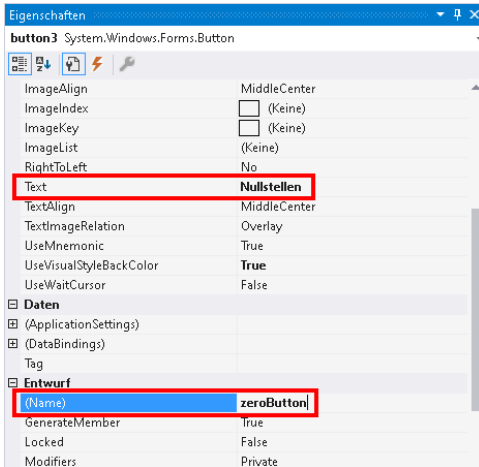


Abb. 7.6 Die Eigenschaften „Text“ und „Name“ der Schaltfläche „Nullstellen“ setzen



Wichtig

Verwechseln Sie nicht Text und Name: Der Text ist die im GUI sichtbare Beschriftung, der Name wird vom Programm für den Zugriff verwendet.

Objekt	Beschriftung (Text)	Name
Button	Mit WTX verbinden	connectButton
	Tarieren	tareButton
	Nullstellen	zeroButton
RadioButton	Netto	netRadioButton
	Brutto	grossRadioButton
TextBox	—	ipTextBox
	—	weightTextBox

7.2.3 Programm-Code schreiben

Öffnen Sie das Fenster für den Programmcode, indem Sie auf eine freie Fläche Ihrer GUI-Form doppelklicken. Ein Teil des Codes, die grundlegende Codestruktur, ist bereits angelegt.

- Fügen Sie die benötigten Komponenten für die HBM Automation API hinzu (*Abb. 7.7*).

Da im Folgenden nur der Jet-Bus verwendet wird, ist die Angabe von WTX.Modbus überflüssig und hier nur der Vollständigkeit halber aufgeführt. Gleiches gilt umgekehrt für WTX.Jet, falls Sie nur eine Verbindung über Modbus verwenden möchten. Für die DSE verwenden Sie DSE.Jet.

- Ändern Sie bei Bedarf die voreingestellten Namen (im **Projektmappen-Explorer** bzw. in den **Eigenschaften**). Im Beispiel wird überall WTX_Demo verwendet, daher auch der Name für den Namespace.

```
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.Automation.Api.Data;
using Hbm.Automation.Api.Weighing.WTX;
using Hbm.Automation.Api.Weighing.WTX.Jet;
using Hbm.Automation.Api.Weighing.WTX.Modbus;

namespace WTX_Demo
{
    public partial class WTX_Demo : Form
    {
        public WTX_Demo()
        {
            InitializeComponent();
        }
    }
}
```

Abb. 7.7 Beginn des Codes für das Programm WTX_Demo (Beispielprogramm)

- Falls eine Funktion Form1_Load bzw. WTX_Demo_Load existiert, können Sie diese löschen, sie wird hier nicht benötigt.
- Fügen Sie den Verweis auf die HBM Automation API hinzu (*Abb. 7.8* und *Abb. 7.9*).

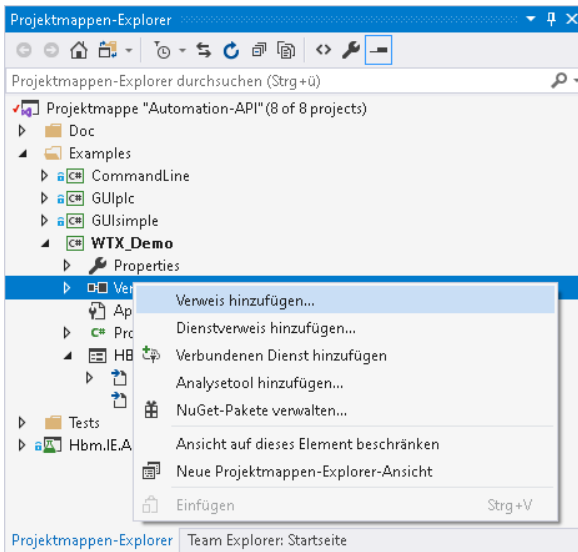


Abb. 7.8 Verweis hinzufügen

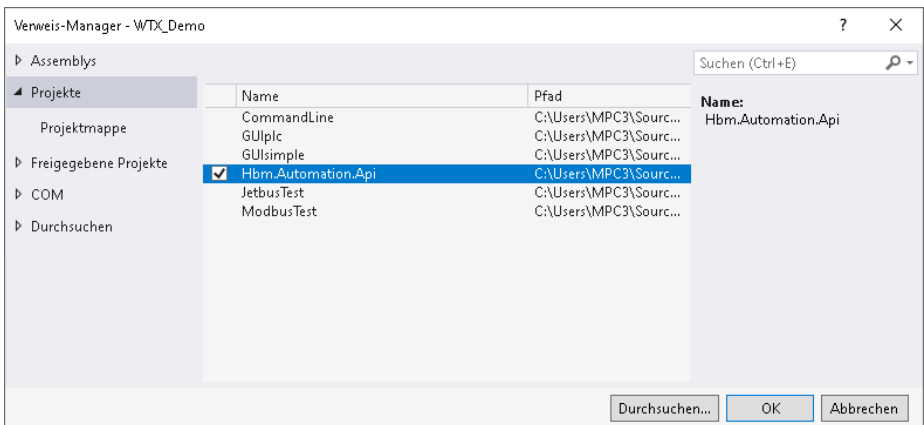


Abb. 7.9 HBM Automation API auswählen

Die Schaltfläche „Mit WTX verbinden“ soll das WTX-Wägeterminal mit dem PC verbinden. Dazu muss im Feld neben „IP-Adresse“ die richtige Adresse eingegeben worden sein, ansonsten wird die voreingestellte Adresse verwendet.

- ▶ Doppelklicken Sie im Entwurfswindow auf die Schaltfläche „Mit WTX verbinden“.

Dadurch wird die Prozedur `connectButton_Click` (Abb. 7.10) angelegt.

```
namespace WTX_Demo
{
    public partial class WTX_Demo : Form
    {
        public WTX_Demo()
        {
            InitializeComponent();
        }

        private void connectButton_Click(object sender, EventArgs e)
        {
        }
    }
}
```

Abb. 7.10 Prozedur für die Schaltfläche „Mit WTX verbinden“ nach dem Anlegen



Information

Falls Sie einen Codeabschnitt löschen, den Sie über einen Doppelklick auf ein Objekt im Entwurfswindow erzeugt haben, erhalten Sie eine Fehlermeldung, wenn Sie das Entwurfswindow erneut öffnen. Da der Code ja noch im Designerfenster (`Resources.Designer`) existiert, müssen Sie auch dort den Code löschen. Am einfachsten geht dies, wenn Sie auf „Gehe zu Code“ in der rechten oberen Ecke der Fehlermeldung klicken.

- ▶ Deklarieren Sie die Objekte `JetBusConnection` und `WTXDevice` für `JetBus` (`WTXJet`) im Kopf der Klasse (Abb. 7.11).
- ▶ Da für die Verbindung die IP-Adresse benötigt wird, benötigen wir außerdem eine Variable für diese Adresse (Abb. 7.11).

```
namespace WTX_Demo
{
    public partial class WTX_Demo : Form
    {
        JetBusConnection _jetConnection;
        WTXJet _wtxDevice;
        string _ipAddress;
    }
}
```

Abb. 7.11 Benötigte Objekte und Variable deklarieren

Neben der Initialisierung muss noch eine Voreinstellung gesetzt werden: welcher Messwert soll in der Voreinstellung angezeigt werden?

- Fügen Sie den benötigten Code nach dem Initialisieren ein (Abb. 7.12).

Damit können Sie jetzt den Code für die Schaltfläche erstellen.

- Lesen Sie die IP-Adresse aus (Abb. 7.12).
- Geben Sie diese für die Verbindung an, indem Sie das Objekt `JetBusConnection` verwenden.
- Dieses Objekt dient im nächsten Schritt als Eingabe für `WTXJet`. Der Parameter 50 ist die Aktualisierungsrate in Millisekunden.

```

namespace WTX_Demo
{
    public partial class WTX_Demo : Form
    {
        JetBusConnection _jetConnection;
        WTXJet _wtxDevice;
        string _ipAddress;

        public WTX_Demo()
        {
            InitializeComponent();
            netRadioButton.Checked = true;
        }

        private void connectButton_Click(object sender, EventArgs e)
        {
            _ipAddress = ipTextBox.Text;
            _jetConnection = new JetBusConnection(_ipAddress, "Administrator", "wtx");
            _wtxDevice = new WTXJet(_jetConnection, 50, update);

            try
            {
                _wtxDevice.Connect(2000);
                MessageBox.Show("WTX verbunden");
            }
            catch (Exception)
            {
                MessageBox.Show("Die Verbindung konnte nicht hergestellt werden");
            }
        }
    }
}

```

Abb. 7.12 Benötigte Objekte und Variable deklarieren

- Um die Verbindung aufzubauen, müssen Sie eine Try-Catch-Konstruktion verwenden, da beim Verbindungsaufbau verschiedene Exceptions auftreten

können. Außerdem müssen Sie einen Timeout angeben, im Beispiel 2000 (ms).

Wie Sie im Beispiel sehen können, gibt es in der Zeile „_wtXDevice =“ drei Parameter für WTXJet. Der dritte Parameter ist hierbei eine Funktion, damit wir eine kontinuierliche Anzeige des Gewichtes erhalten. Die Funktion update soll abhängig von den RadioButtons kontinuierlich den Netto- oder Bruttowert vom WTX-Wägeterminal auslesen. Den Code für diese Funktion sehen Sie in *Abb. 7.13*.

```
private void update(object sender, ProcessDataReceivedEventArgs e)
{
    string weight;

    weightTextBox.BeginInvoke(new Action(() =>
    {
        if (this.netRadioButton.Checked)
            weight = e.ProcessData.PrintableWeight.Net;
        else
            weight = e.ProcessData.PrintableWeight.Gross;
        weightTextBox.Text = weight + " " + _wtXDevice.Unit;
    }));

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

Abb. 7.13 Funktion update

- Erstellen Sie zuletzt den Code für die beiden Schaltflächen **Nullstellen** und **Tarieren**.


```

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

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

Abb. 7.14 Die Funktionen für die beiden Schaltflächen „Tarieren“ und „Nullstellen“

Im Beispiel wurde noch über **Eigenschaften** die **PictureBox** mit dem HBM-Logo belegt (Abb. 7.15). Klicken Sie dazu auf das Symbol mit den drei Punkten in der Zeile **Image** und importieren Sie ein Bild Ihrer Wahl als **Lokale Resource**.

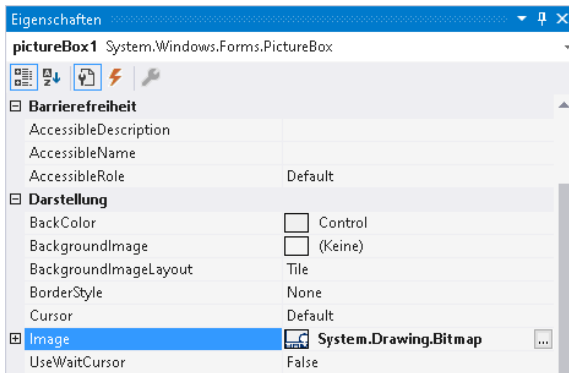


Abb. 7.15 Bild laden

Damit ist das Programm fertig. Wählen Sie es oben im Fenster aus und starten Sie es. Falls Sie sich nirgends vertippt haben, wird das Programm erstellt und gestartet (Abb. 7.16).

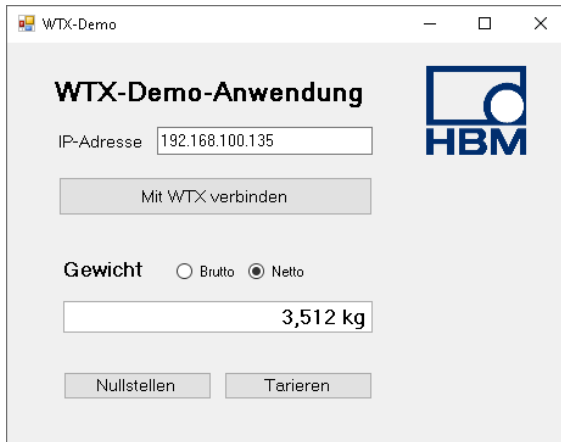


Abb. 7.16 Das fertige Programm

Nachfolgend noch einmal (auf zwei Seiten) das komplette Programm. Automatisch eingefügte Zeilen, die unnötig sind, wurden hier entfernt, um das Programm auf die wesentlichen Zeilen zu konzentrieren.

```
using System;
using System.Windows.Forms;
using Hbm.Automation.Api.Data;
using Hbm.Automation.Api.Weighing.WTX;
using Hbm.Automation.Api.Weighing.WTX.Jet;
namespace WTX_Demo
{
    public partial class WTX_Demo : Form
    {
        JetBusConnection _jetConnection;
        WTXJet _wtxDevice;
        string _ipAddress;

        public WTX_Demo()
        {
            InitializeComponent();
            netRadioButton.Checked = true;
        }
    }
}
```

```

private void connectButton_Click(object sender, EventArgs e)
{
    _ipAddress = ipTextBox.Text;
    _jetConnection = new JetBusConnection(_ipAddress, "Administrator", "wtx");
    _wtxDevice = new WTXJet(_jetConnection, 50, update);
    try
    {
        _wtxDevice.Connect(2000);
        MessageBox.Show("WTX verbunden");
    }
    catch (Exception)
    {
        MessageBox.Show("Die Verbindung konnte nicht hergestellt werden");
    }
}

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

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

private void update(object sender, ProcessDataReceivedEventArgs e)
{
    string weight;

    weightTextBox.BeginInvoke(new Action(() =>
    {
        if (this.netRadioButton.Checked)
            weight = e.ProcessData.PrintableWeight.Net;
        else
            weight = e.ProcessData.PrintableWeight.Gross;
        weightTextBox.Text = weight + " " + _wtxDevice.Unit;
    }));

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

```

8 FAQ: Häufig gestellte Fragen

Was ist Open Source?

Open-Source-Software ist Software, deren Quelltext offen gelegt wurde und die meist kostenlos genutzt werden kann. Üblicherweise darf sie auch verändert werden, hier gibt es aber in einigen Fällen bestimmte Auflagen zu beachten. Die HBM Automation API dürfen Sie sowohl verändern als auch kostenlos weitergeben, es werden keine Lizenzgebühren oder ähnliches fällig und es besteht nur die Auflage, den Urheberrechtshinweis (Copyright) und den Genehmigungshinweis in Ihrer Software zu verwenden (siehe GitHub).

Was ist eine API?

Eine API (Application Programming Interface) ist eine Programmierschnittstelle zur leichten Erstellung von Anwendungsprogrammen. Es ist quasi eine Sammlung von Funktionen, mit denen Sie die Kommunikation zwischen verschiedenen Komponenten eines Systems ermöglichen, ohne die konkret zu verwendenden Anweisungen kennen zu müssen. Die API stellt damit alle Bausteine zur Verfügung, die Sie bei der Entwicklung eines Anwenderprogramms benötigen, um mit den Geräten zu kommunizieren.

Was ist der Unterschied zwischen Modbus und Jet-Bus?

Modbus ist ein serielles Kommunikationsprotokoll und gilt als Standard für die Kommunikation mit einer SPS (Speicherprogrammierbare Steuerung). Das Modbus-Protokoll verwendet standardmäßig den TCP-Port 502.

Jet bzw. der Jet-Bus (<https://jetbus.io>) ist ein Framework (Programmiergerüst), das für schnelle Netzwerkkommunikation entwickelt wurde, speziell für IoT (Internet of Things). Es verwendet ein Publish/Subscribe-Messaging, das mehreren Clients die Verbindung zu einem Server ermöglicht. Die Jet-Kommunikation basiert auf dem TCP-basierten WebSocket-Protokoll unter Verwendung des TCP-Ports 443.

Was ist ein WebSocket?

WebSocket ist ein Kommunikationsprotokoll, das für den Betrieb über den TCP-Port 443 ausgelegt ist. Es ermöglicht eine bidirektionale Kommunikation über eine (einzige) TCP-Verbindung und damit die interaktive Kommunikation

zwischen einer Webanwendung (Browser) und dem WebSocket-Server. Darüber hinaus werden sichere (verschlüsselte) Verbindungen unterstützt.

Was ist der Unterschied zwischen der HBM Automation API und normalen industriellen Schnittstellen, z. B. für SPS?

Die HBM Automation API nutzt Jet, um Ihnen die effiziente Erstellung eigener Lösungen über eine 2-Wege-verschlüsselte TCP/IP-Verbindung zu ermöglichen. Die benötigten Programmierbausteine für die Kommunikation sind in der API bereits vorhanden.

SPS-Schnittstellen wie PROFINET[®], EtherNet/IP[™] oder PROFIBUS[®] stellen lediglich einen Standard für die Feldbuskommunikation dar. Die Programmierung der Kommunikation über diese Schnittstellen muss also erst erfolgen.

Was bedeutet eine 2-Wege-Verschlüsselung über https?

Eine einseitige (1-Weg-) Verschlüsselung bedeutet, dass ein System nur verschlüsseln kann. Ein Beispiel dafür ist die Eingabe eines Passworts, das verschlüsselt und dann mit der Authentifizierung verglichen werden muss – eine Entschlüsselung ist hier nicht notwendig. Für die sichere Übertragung von verschlüsselten (Mess-)Daten ist aber auch eine Entschlüsselung notwendig, um die übertragenen Daten wieder lesbar zu machen. Dies wird als 2-Wege-Verschlüsselung bezeichnet und setzt voraus, dass alle Netzwerkteilnehmer über eine Verschlüsselungs- und eine Entschlüsselungsfunktion verfügen, um eine sichere bidirektionale Kommunikation zu ermöglichen. Die Verschlüsselung der HBM Automation API basiert auf dem HTTPS-Kommunikationsprotokoll und verwendet das kryptographische Protokoll SSL. Damit werden die Netzwerkteilnehmer mithilfe eines Handshake-Mechanismus authentifiziert, um sich vor Lauschangriffen zu schützen.

9 Technischer Support

Sollten bei der Arbeit mit der HBM Automation API Probleme auftreten, können Sie folgende Dienste nutzen:

E-Mail-Unterstützung

support@hbkworld.com

Telefon-Unterstützung

Die telefonische Unterstützung ist von 9:00 bis 17:00 Uhr (MEZ bzw. MESZ) an allen Werktagen verfügbar: +49 6151 803-0

Folgende Möglichkeiten stehen Ihnen ebenfalls zur Verfügung: HBM-Support und Vertrieb weltweit: <https://www.hbm.com/en/0051/worldwide-contacts/>.

Hauptsitze weltweit

Europa:

Hottinger Brüel & Kjaer GmbH

Im Tiefen See 45, 64293 Darmstadt, Deutschland

Nord- und Südamerika:

HBM Inc., 19 Bartlett Street, Marlborough, MA 01752, USA

Tel. +1 800-578-4260 / +1 508-624-4500

Fax +1 508-485-7480

E-Mail: info@usa.hbm.com

Asien:

Hottinger Baldwin Measurement (Suzhou) Co., Ltd.

106 Heng Shan Road, Suzhou 215009, Jiangsu, VR China

Tel. +86 512-68247776, Fax +86 512-68259343

E-Mail: hbmchina@hbm.com.cn

HBM Test and Measurement

Tel. +49 6151 803-0

Fax +49 6151 803-9100

info@hbm.com

measure and predict with confidence



A05600_01_X00_00 HBM: public

www.hbm.com