PLC and PC
communication via SLMP Protocol

Mitsubishi Electric Europe

Revisions

Date	Version	Revision
June, 2020	V001	First edition
November, 2020	V002	Modyfied of library

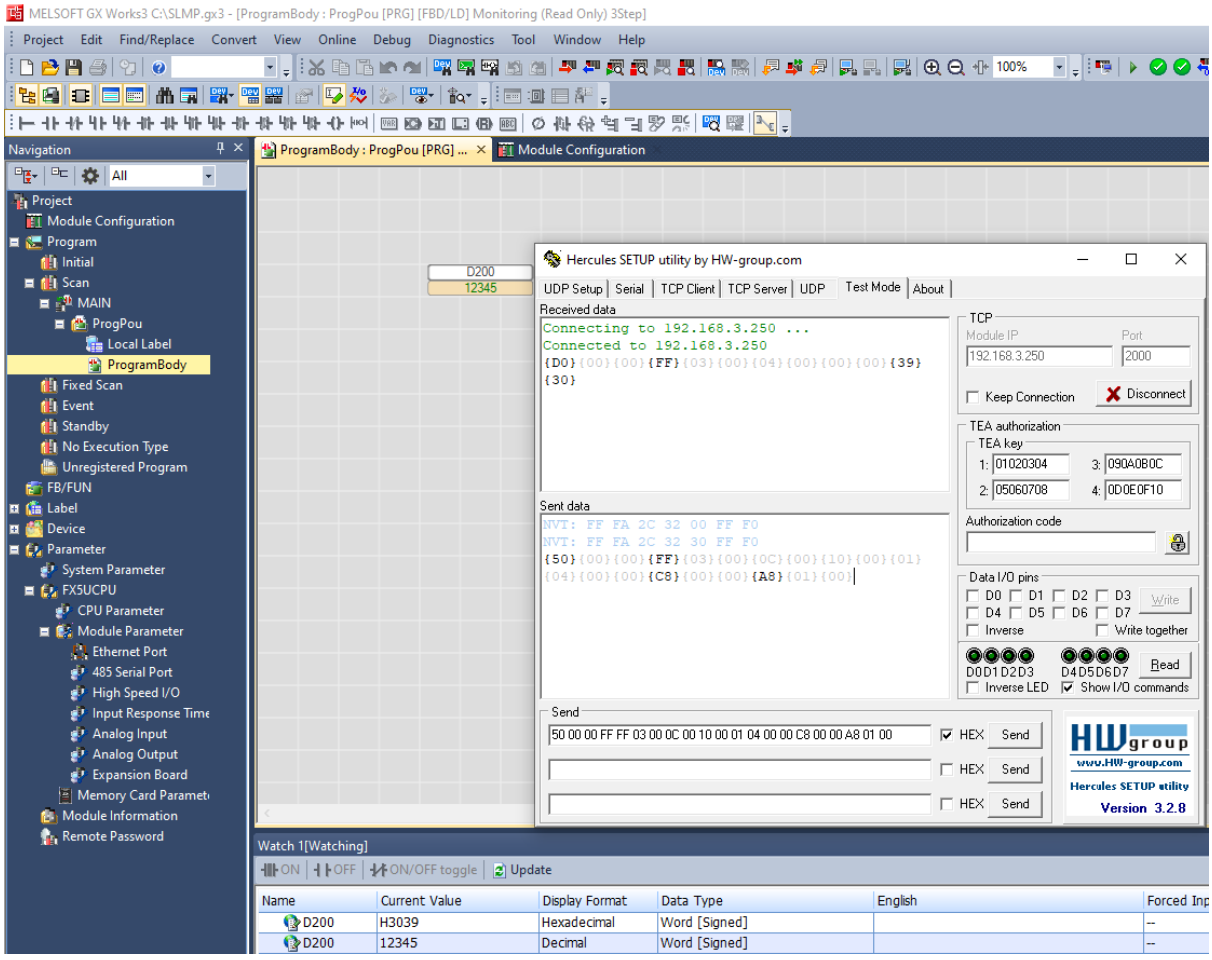
Table of contents

1. Overview of document

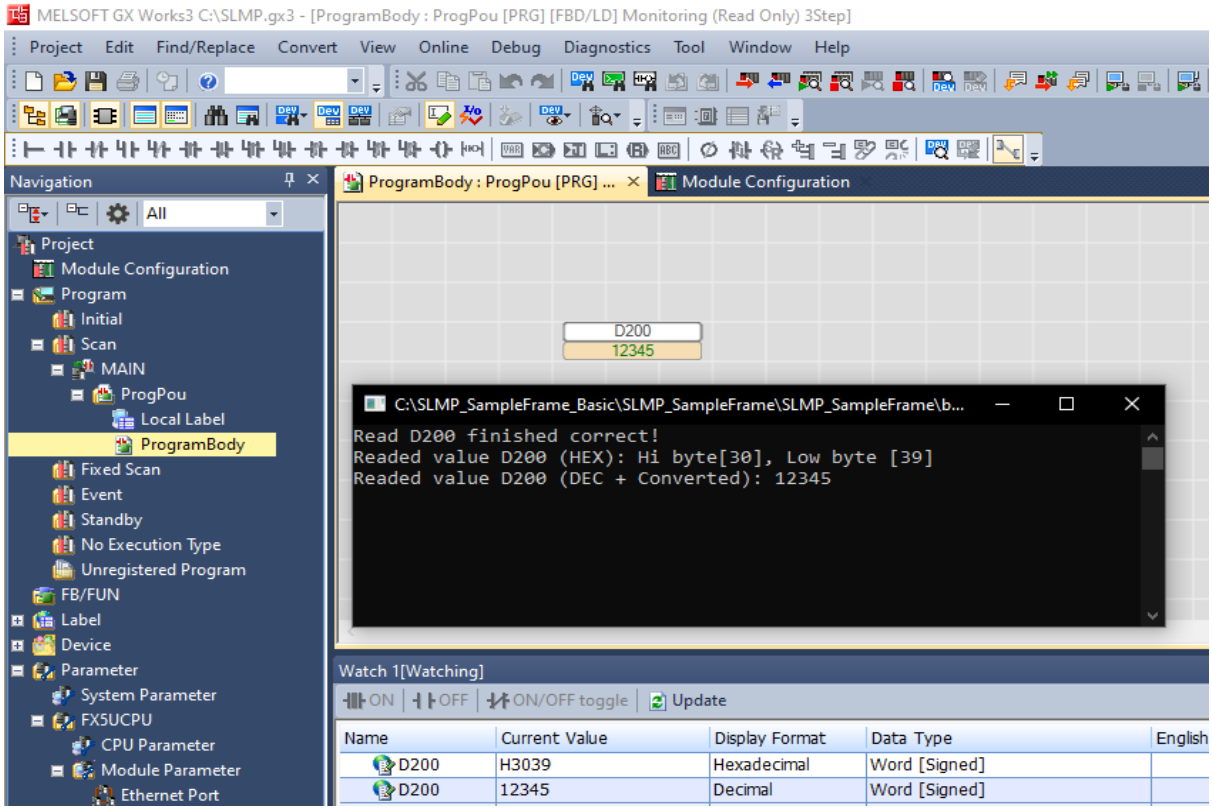
1.1. Overview of PC and PLC communication via SLMP

The purpose of this document is to provide simple solution for PC and PLC communication via SLMP. A the beginning, there will be presented how to set communication parameters in GX Works3 for SLMP protocol. Then it will be show how to read a registry value from PLC using Hercules SETUP utility program. In addition, presented document contains information how to create your own console application in C# language.

1.2. Example effects



SLMP communication using Hercules SETUP utility



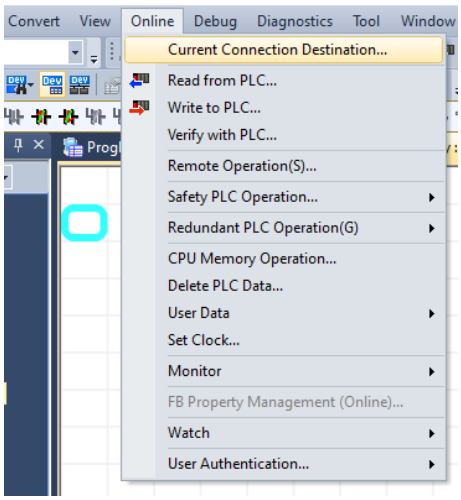
SLMP communication via Console Application

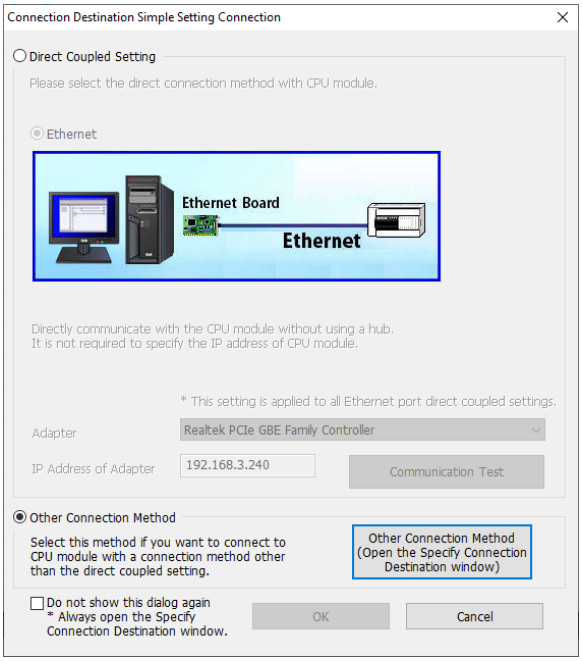
2. Details of PC and PLC communication via SLMP

2.1. Parametrization in GX Works3

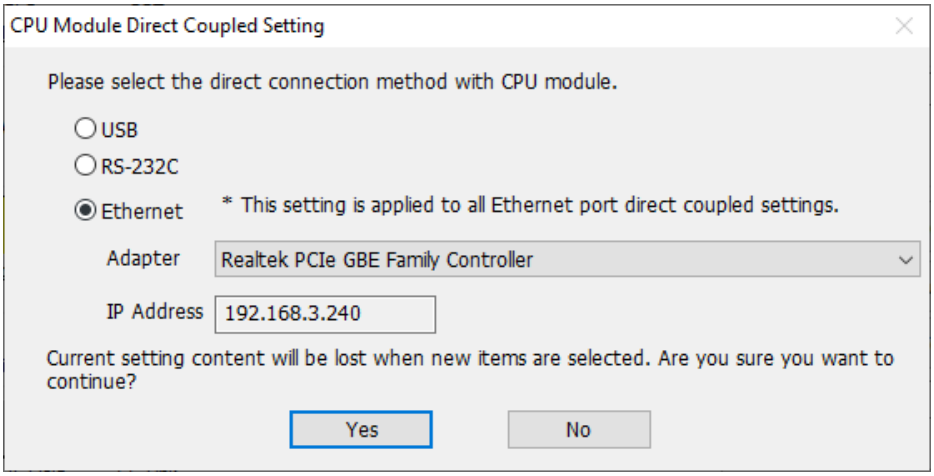
PC and PLC connection

To establish connection between PC and PLC: **Connect devices** with Ethernet wire → **Create new project** for your PLC select → **Select** Online → Current Connection Destination…



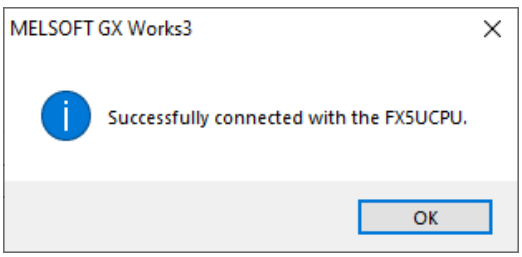


For **Direct Coupled Setting** we need only to **put IP Address of Adapter** and confirm settings (if communication test was positive).
Otherwise **select Other Connection Method** (Open the specify Connection Destiny window) → **CPU Module Direct Couple Setting** → select **Ethernet** → put **IP Address** → **Yes**



Remember to set IP Address of your PC in the same subnet as the PLC.

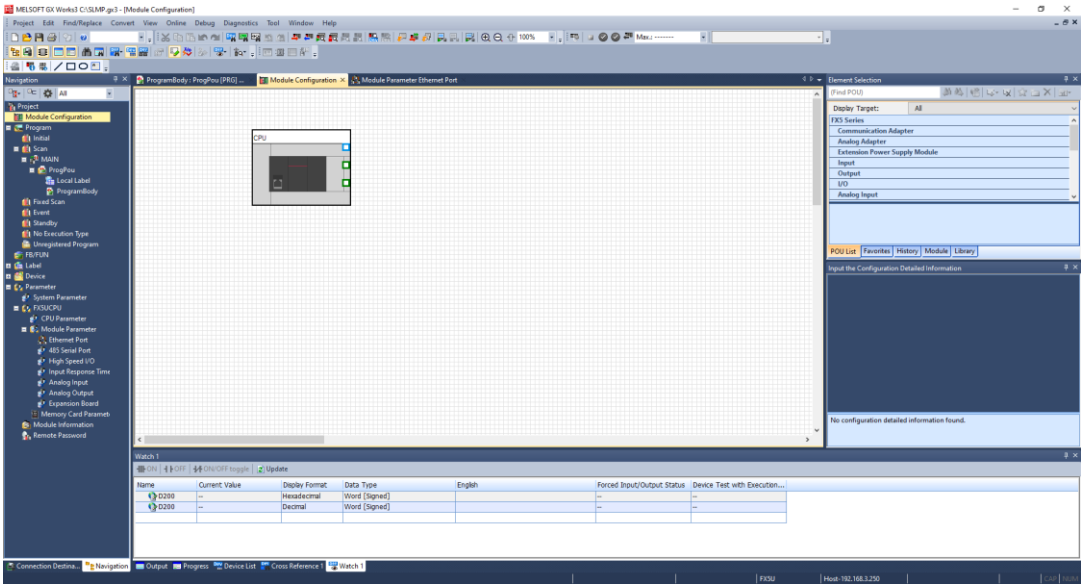
Execute **Communication Test** and confirm the settings .



Module Configuration

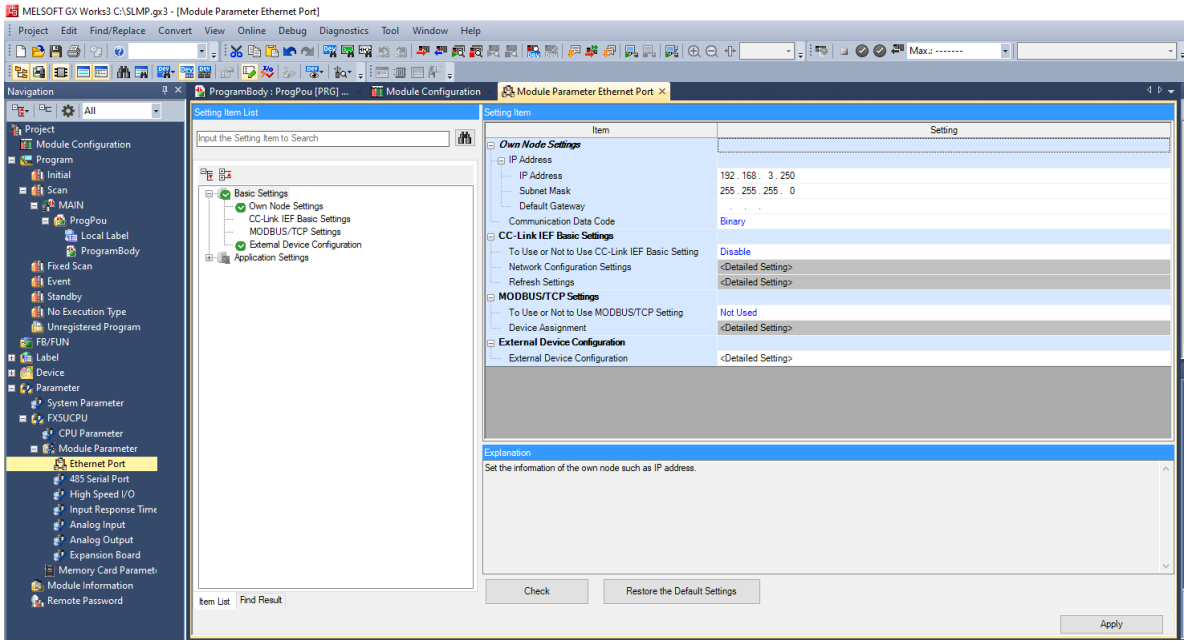
Double click on **Module Configuration** from the project tree → by drag and drop **place hardware components** that represent the actual state.

You can also select **Online** → **Read Module Configuration from PLC**.

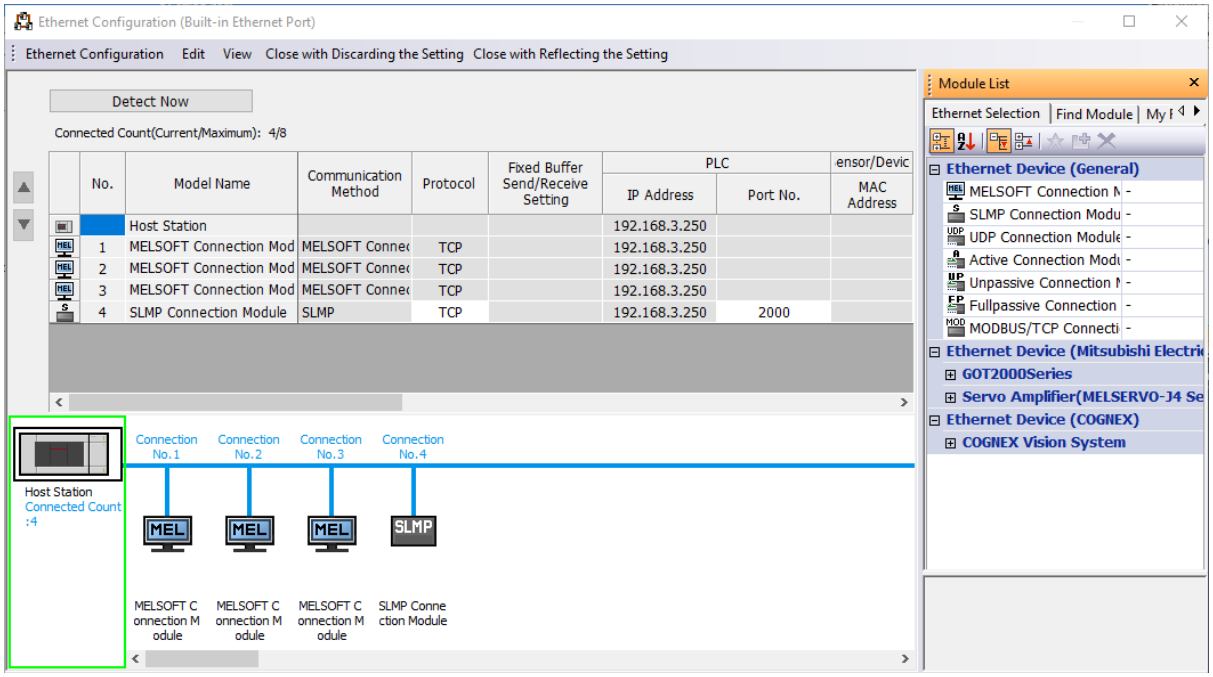


Ethernet Port

To start using SLMP Protocol set parameters as follow:



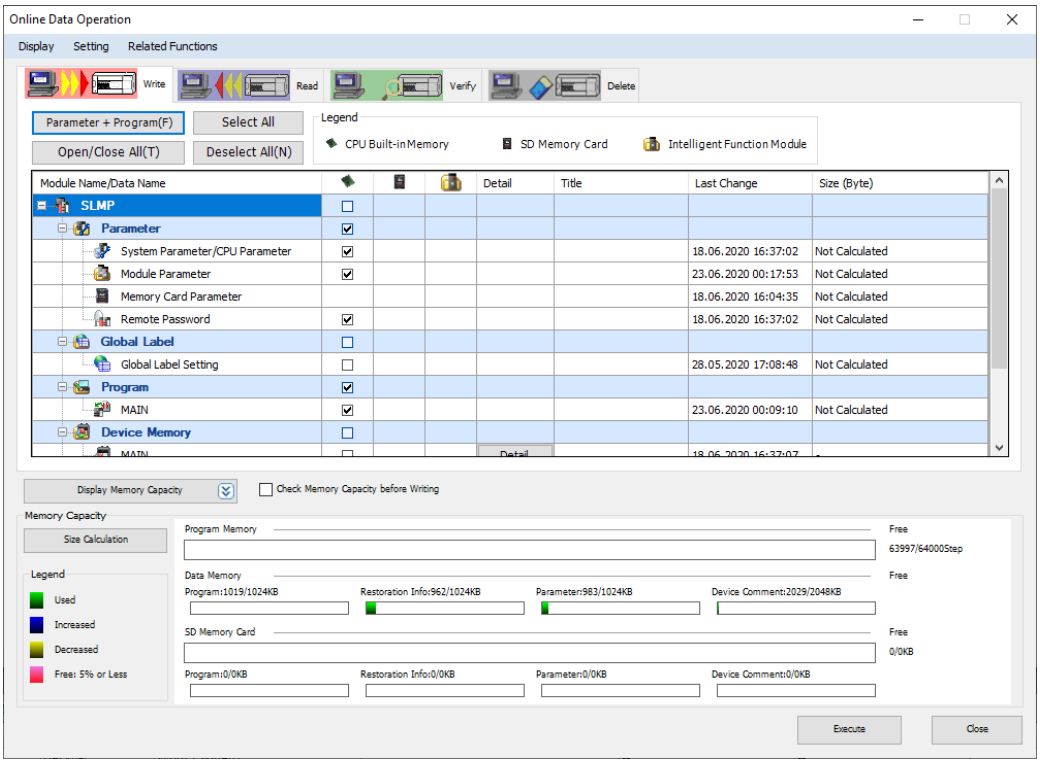
Set **External Device Configuration** by double click on **<Detailed Settings>** next to this parameter.
Add SLMP Connection Module from **Ethernet Devices** by drag and drop it → **set TCP** protocol → **set Port No.** on **2000** → **Close with Reflecting the Setting**.



After this operations select **Check** and **Apply**.

Write to PLC...

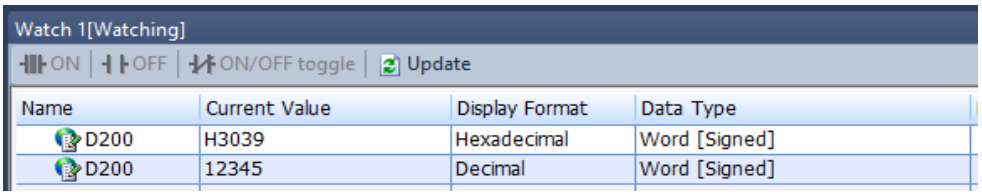
To write your project to PLC **select Online →click Write to PLC... → select Parameter + Program(F) → Execute**



Reset PLC and go to **RUN** mode.

Monitor D200 parameter

Add parameter D200 twice to the **Watch window** by clicking on the row and putting "**D200**". For the first case choose **Decimal** display format, and **Hexadecimal** for second one. To monitor and change current value click on **Start Monitoring** .

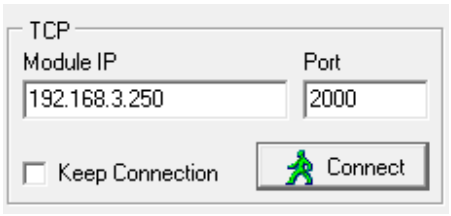


2.2. Hercules SETUP utility

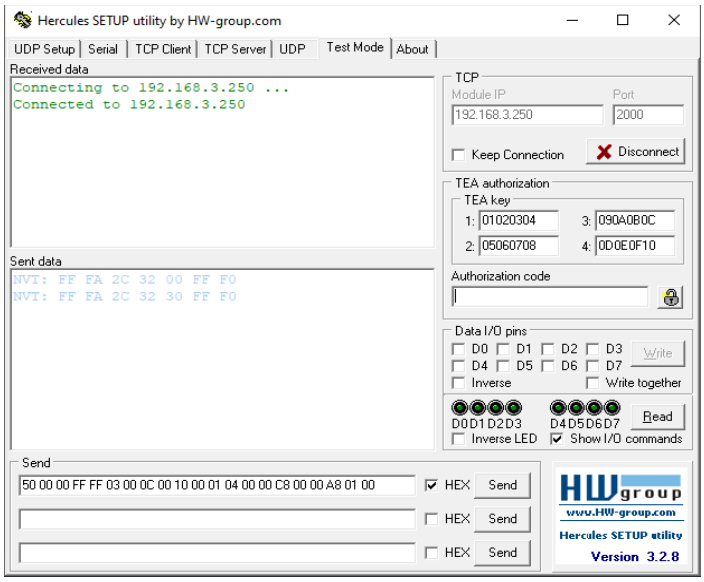
Connection with PLC

Open Hercules SETUP utility application → select **Test mode** → set **Module IP: 192.168.3.250** (IP address of PLC set in GX Works3) → set **Port: 2000** (as in Ethernet Configuration in GX Works3)

If you don't have the application installed yet, here there is a link to the website with the **software**: https://www.hw-group.com/software/hercules-setup-utility



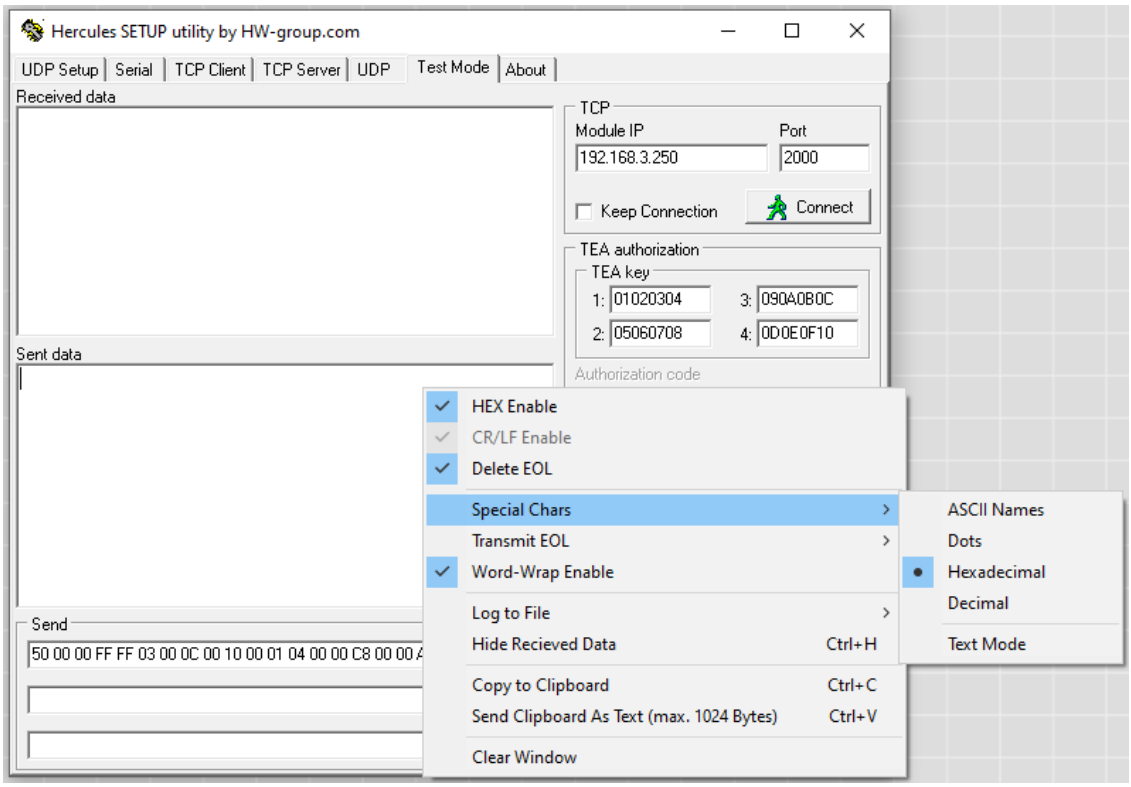
Make sure that IP address and Port No. are the same as those set in GX Works3.
After this operation you should be able to start communication. To do this, **press Connect**.



Connection operation status will be displayed in **Received data: Connected to 192.168.3.250** (PLC address).

Sending request to PLC

Before you will send request to PLC, make sure the settings look like these:



Put the request into **Send** section:

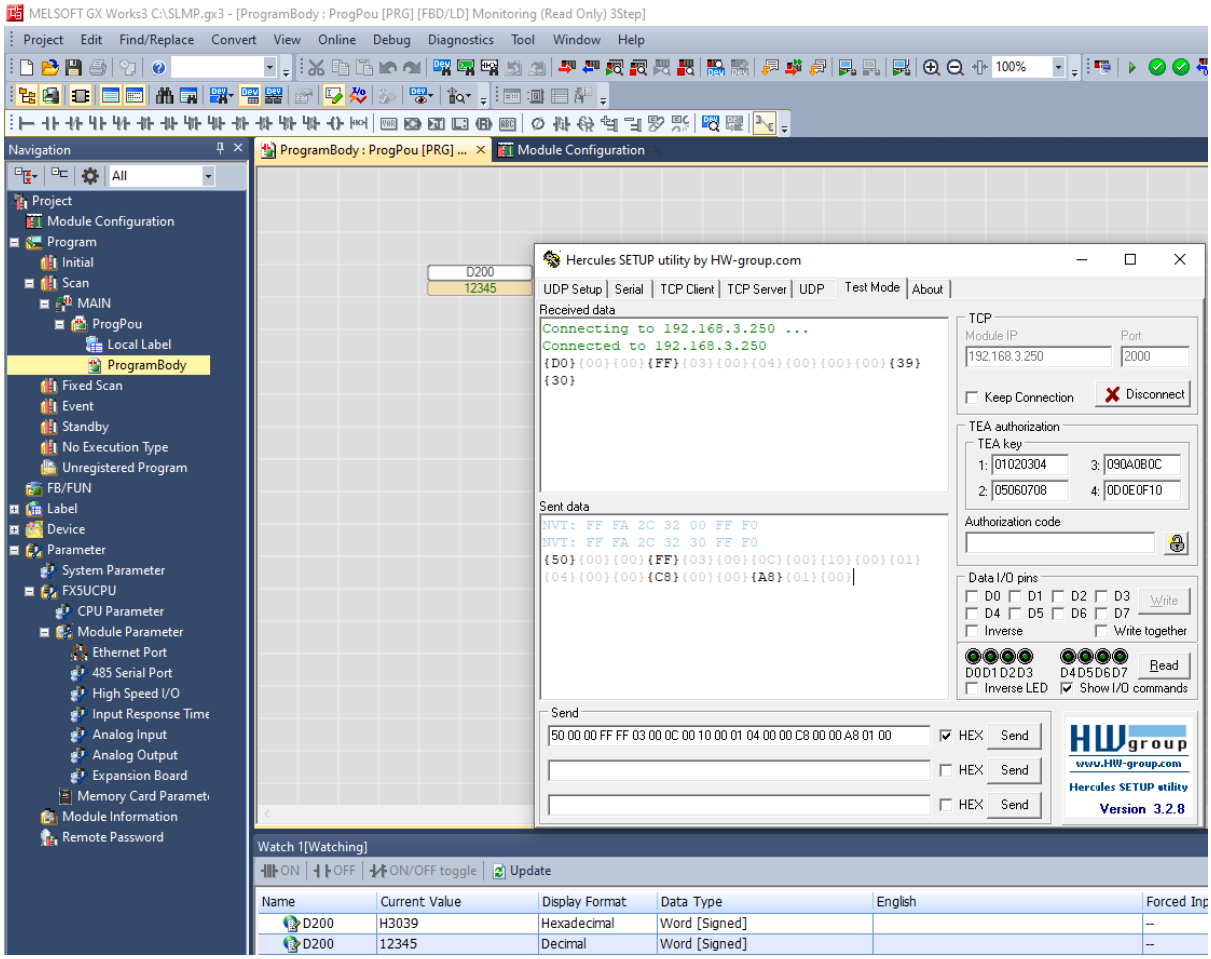
“50 00 00 FF FF 03 00 0C 00 10 00 01 04 00 00 C8 00 00 A8 01 00”

Which is synonymous with:

Request	**50**	**00**	**00**	**FF**	**FF**	**03**	**00**	**0C**	**00**
	Subheader (without serial No.)		**Request destination network No.**	**Request destination station No.**	**Request destination module I/O No.**		**Request destination multidrop station No.**	**Request data length**	
Response	**D0**	**00**	**00**	**FF**	**FF**	**03**	**00**	**04**	**00**

10	**00**	**01**	**04**	**00**	**00**	**C8**	**00**	**00**	**A8**	**01**	**00**
Monitoring timer		**Comand (0401:Read)**		**Subcommand**		**Head Device No. (0xC8 => 200DEC)**			**Device code (D register)**	**No. of device points**	
00	**00**	**BE**	**DC**								
Error code		**Response data**									

Select **Hex** option and click **Send.**
In presented situation, sent command asks PLC about **D200 register**.
All sent commands are displayed in the **Sent data** section.



Data from the PLC are received and displayed in hexadecimal form in **Received data** section.
Observing this register in **Watch window** in GX Works3 you can confirm that data was sent properly.

2.3. C# Console Application

Run ready application

Write the IP address of PLC into **byAdres[]** as shown below → put **ipAdress** and **2000** (Port No. - value as in GX Works3 settings) as arguments of **ConnectTCP** function

```
using System;
using System.Net.Sockets;
using System.Net;
using System.Net.NetworkInformation;

namespace SLMP_SampleFrame
{
    class Program
    {
        static TcpClient tcpC = new TcpClient(); // Global TcpClient object
        static void Main(string[] args)
        {
            byte[] byAdres = new byte[4];
            // set IP address of PLC
            byAdres[0] = 192; byAdres[1] = 168; byAdres[2] = 3; byAdres[3] = 250;
            IPAddress ipAdress = new IPAddress(byAdres);
           ConnectTCP(ipAdress, 2000); //connection for set IP address and Port No.
            Read_D200(); // read D200 register
            Console.ReadKey();
        }

    }
}
```

Main program

In "**Part of code for read D200 register**" section, put the request into **payload** variable:

"0x50, 0x00, 0x00, 0xff, 0xff, 0x03, 0x00, 0x0C, 0x00, 0x10, 0x00, 0x01, 0x04, 0x00, 0x00, 0xC8, 0x00, 0x00, 0xA8, 0x01, 0x00"

Which is synonymous with:

Request	**50**	**00**	**00**	**FF**	**FF**	**03**	**00**	**0C**	**00**
	Subheader (without serial No.)		**Request destination network No.**	**Request destination station No.**	**Request destination module I/O No.**		**Request destination multidrop station No.**	**Request data length**	
Response	**D0**	**00**	**00**	**FF**	**FF**	**03**	**00**	**04**	**00**

10	00	01	04	00	00	C8	00	00	A8	01	00
Monitoring timer		Comand (0401:Read)		Subcommand		Head Device No. (0xC8 => 200DEC)			Device code (D register)	No. of device points	

00	00	BE	DC
Error code		Response data	

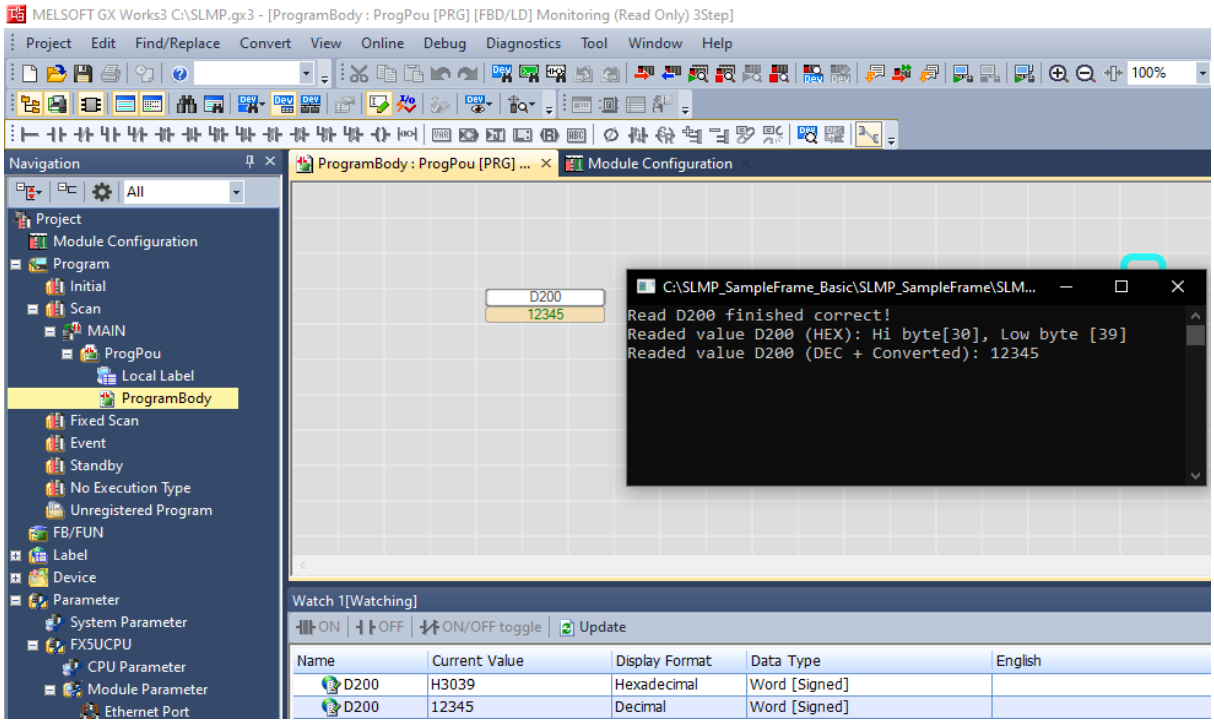
Each byte in created table is hexadecimal and separated by a comma.
In presented situation, sent command asks PLC about **D200 register**.

```
#region Part of code for read D200 register
        static void Read_D200()
        {
            //Request frame for read D200 register
            byte[] payload = new byte[] { 0x50, 0x00, 0x00, 0xff, 0xff, 0x03, 0x00,
0x0C, 0x00, 0x10, 0x00, 0x01, 0x04, 0x00, 0x00, 0xC8, 0x00, 0x00, 0xA8, 0x01, 0x00 };
            NetworkStream stream = tcpC.GetStream();
            stream.Write(payload, 0, payload.Length);
            byte[] data = new Byte[20];
            stream.ReadTimeout = 1000;
            try
            {
                Int32 bytes = stream.Read(data, 0, data.Length);
                if (data[9] == 0 && data[10] == 0)
                {
                    byte lowbyteResponse = data[11];
                    int hibyteResponse = data[12];
                    int afterConversion = (hibyteResponse << 8) + lowbyteResponse;

                    // Show information about D200 register and operation status
                    Console.WriteLine("Read D200 finished correct!");
                    Console.WriteLine("Readed value D200 (HEX): Hi byte[" +
hibyteResponse.ToString("X") + "], Low byte [" + lowbyteResponse.ToString("X") + "]");
                    Console.WriteLine("Readed value D200 (DEC + Converted): " +
afterConversion.ToString());
                }
                else
                {
                    Console.WriteLine("Error in Answere");
                }
            }
            catch
            {
                Console.WriteLine("Error in interpreter");
            }
        }
        #endregion
```

Part of code for read D200 register

Launch your application.



Data from the PLC are received and displayed in console. Firstly in **hexadecimal form**, detail **Low** and **High byte**. Secondly this bytes are converted and displayed in **decimal form**.
Before displaying register value, information about operation status is shown.
Observing this register in Watch window in GX Works3 you can confirm that data was sent properly.

Caution!

Presented code is only a sample program, so it should not be run directly on a real object!
Customize the created application to your needs.
The number of sent and receive data can vary. In presented example, the number of frame elements is static.

3. Modifications

3.1. MakePingTest

Adding this function will make establishing a connection easier. User gets information whether the device responds.

```
#region Function for perform Ping test with real PLC
        static bool MakePingTest(IPAddress IPAddressForTest)
        {
            bool pingAns = false;
            Ping pingSender = new Ping();
            PingReply reply = pingSender.Send(IPAddressForTest);
            if (reply.Status == IPStatus.Success)
            {
                pingAns = true;
            }
            return pingAns;
        }
        #endregion
```

Function for perform Ping test with real PLC

3.2. SelfTest

You can also add function that will check the correctness of the sent data frame. This function compares sent and received data (in this case the test is performed on a 5-element set). If data match, then data can be exchanged between devices and communication starts.

```
        #region Part of code used to verify whether the communication function operates
normally or not
        static bool SelfTest()
        {
            bool loopTestAns = false;

            byte[] loopMessage = new byte[5] {0x41, 0x42, 0x43,0x44, 0x45}; // 5 elements
for test - "ABCDE"


            //Request data length
            int needByteMessage = 2 + 4 + 2 + loopMessage.Length;
            byte lowByte = (byte)(needByteMessage & 0xff);
            byte highByte = (byte)(needByteMessage >> 8 & 0xff);


            byte[] payload = new byte[] { 0x50, 0x00, 0x00, 0xff, 0xff, 0x03, 0x00,
lowByte, highByte, 0x10, 0x00, 0x19, 0x06,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00 };

            //number of loopack data
            lowByte = (byte)(loopMessage.Length & 0xff);
            highByte = (byte)(loopMessage.Length >> 8 & 0xff);
            payload[15] = lowByte; payload[16] = highByte;

            // loopack data
            for (int i = 0; i < loopMessage.Length; i++)
            {
```

```
                payload[17 + i] = loopMessage[i];
            }

            NetworkStream stream = tcpC.GetStream();
            stream.Write(payload, 0, payload.Length);
            byte[] data = new Byte[20];
            stream.ReadTimeout = 1000;
            try
            {
                Int32 bytes = stream.Read(data, 0, data.Length);
                if (data[9] == 0 && data[10] == 0 && data[11] == lowByte && data[12] ==
highByte)
                {
                    loopTestAns = true;
                    for (int i = 0; i < loopMessage.Length; i++)
                    {
                        if (loopMessage[i] != data[13 + i])
                        {
                            loopTestAns = false;
                        }
                    }
                }
            }
            catch
            {
                loopTestAns = false;
            }
            return loopTestAns;
        }
        #endregion
```

Function for check the correctness of the sent data frame

3.3. Simple library

It is possible to use simple library. All data you can find in attached file: „Simple Library for SLMP communication.zip”. Inside you can find necessary DLL file to import your project and sample program where this library was used.