ClearLastError() public method

Sets the LastErrorCode to NoError and LastErrorString to String.Empty

public **ClearLastError** ( ) : void

return	void

```
public void ClearLastError()
{
    LastErrorCode = ErrorCode.NoError;
    LastErrorString = string.Empty;
}
```

Close() public method

Disonnects from the plc and close the socket

public **Close** ( ) : void

return	void

Plc Class Documentation

EXAMPLE #1

0

Show file

```
public override void DisConnect()
{
    if (_plcReader != null && _plcWriter != null)
    {
        _plcReader.Close();
        _plcWriter.Close();
        // _plcReadTimer.Stop();
    }
    ComState = ComState.DisConnected;
    ComState = ComState.DisConnected;
}
```

EXAMPLE #2

0

Show file

File: Main.cs **Project:** gilpark/qb_challenge

```
        static void Main(string[] args)
        {
            S7.Net.Plc plc     = new Plc(cpu: CpuType.S71200, ip: "10.1.10.142",
rack: 0, slot: 1);
            ErrorCode  errCode = plc.Open();

            var       b1    = (UInt16)plc.Read("MW100");
            ErrorCode write = plc.Write("MW102", 100);

            plc.Close();

            System.Console.WriteLine(b1);
            Console.ReadLine();
        }
```

1. Example of a PLC Latching Program -...

EXAMPLE #3

0

Show file

```
        //public string LastError { get; private set; }
        public override async Task ConnectAsync()
        {
            ComState = ComState.Connecting;

            await DelayAsync(1000);

            S7.Net.CpuType S7NetCpuType = ConvertCpuType(Config.CpuType);

            try
            {
                string ip = "100.67.165.113";
                _plcReader = new Plc(S7NetCpuType, ip /*Config.Ip*/,
(short)Config.Rack, (short)Config.Slot);
                _plcWriter = new Plc(S7NetCpuType, ip /*Config.Ip*/,
(short)Config.Rack, (short)Config.Slot);
                await _plcReader.OpenAsync();

                await _plcWriter.OpenAsync();

                if (_plcReader.IsConnected && _plcWriter.IsConnected)
                {
                    ComState = ComState.Connected;
                }
                else
                {
                    ComState = ComState.ConnectFailed;
                    if (_plcReader.IsConnected)
                    {
                        _plcReader.Close();
                    }
                    if (_plcWriter.IsConnected)
                    {
                        _plcWriter.Close();
                    }
                }
            }
            catch (Exception)
            {
                //LastError = ex.Message;
                ComState = ComState.ConnectFailed;
            }
        }
```

CreateReadDataRequestPackage() private method

Create the bytes-package to request data from the plc. You have to specify the memory type (dataType), the address of the memory, the address of the byte and the bytes count.

```
private CreateReadDataRequestPackage ( DataType dataType, int db, int startByteAdr, int count = 1 ) : Byte
```

```
        private Types.ByteArray CreateReadDataRequestPackage(DataType dataType,
int db, int startByteAdr, int count = 1)
        {
            //single data req = 12
            var package = new Types.ByteArray(12);
            package.Add(new byte[] { 0x12, 0x0a, 0x10 });
            switch (dataType)
            {
                case DataType.Timer:
                case DataType.Counter:
                    package.Add((byte)dataType);
                    break;
                default:
                    package.Add(0x02);
                    break;
            }

            package.Add(Types.Word.ToByteArray((ushort)(count)));
            package.Add(Types.Word.ToByteArray((ushort)(db)));
            package.Add((byte)dataType);
            var overflow = (int)(startByteAdr * 8 / 0xffffU); // handles words
with address bigger than 8191
            package.Add((byte)overflow);
            switch (dataType)
            {
                case DataType.Timer:
                case DataType.Counter:
                    package.Add(Types.Word.ToByteArray((ushort)(startByteAdr)));
                    break;
                default:
                    package.Add(Types.Word.ToByteArray((ushort)((startByteAdr) *
8)));
                    break;
            }

            return package;
        }
```

Dispose() public method

Releases all resources, disonnects from the plc and closes the socket

public **Dispose** ( ) : void

return	void

```
public void Dispose()
{
    if (_mSocket != null)
    {
        if (_mSocket.Connected)
        {
            _mSocket.Shutdown(SocketShutdown.Both);
            _mSocket.Close();
        }
    }
}
```

Open() public method

Open a socket and connects to the plc, sending all the corrected package and returning if the connection was successful (ErroreCode.NoError) of it was wrong.

public **Open** ( ) : ErrorCode

return	ErrorCode

Plc Class Documentation

EXAMPLE #1

0

Show file

File: S7NetTests.cs **Project:** Katsarakis/s7netplus

```
 /// <summary>
 /// Create a plc that will connect to localhost (Snap 7 server) and connect to
it
 /// </summary>
 public S7NetTests()
 {
     plc = new Plc(CpuType.S7300, "127.0.0.1", 0, 2);
     //ConsoleManager.Show();
     ShutDownServiceS7oiehsx64();
     S7TestServer.Start();
     plc.Open();
 }
```

Example of a PLC Latching Program - Schneider Course

EXAMPLE #2

0

Show file

File: Main.cs **Project:** gilpark/qb_challenge

```
        static void Main(string[] args)
        {
            S7.Net.Plc plc     = new Plc(cpu: CpuType.S71200, ip: "10.1.10.142",
rack: 0, slot: 1);
            ErrorCode  errCode = plc.Open();

            var       b1    = (UInt16)plc.Read("MW100");
            ErrorCode write = plc.Write("MW102", 100);

            plc.Close();

            System.Console.WriteLine(b1);
            Console.ReadLine();
```

```
        }
//////////////////////////
//////////////
/

    public ErrorCode Open()
        {
            byte[] bReceive = new byte[256];

            try
            {
                // check if available
                if (!IsAvailable)
                {
                    throw new Exception();
                }
            }
            catch
            {
                LastErrorCode = ErrorCode.IPAddressNotAvailable;
                LastErrorString = string.Format("Destination IP-Address '{0}' is
not available!", IP);
                return LastErrorCode;
            }

            try
            {
                _mSocket = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
                _mSocket.SetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.ReceiveTimeout, 1000);
                _mSocket.SetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.SendTimeout, 1000);
                IPEndPoint server = new IPEndPoint(IPAddress.Parse(IP), 102);
                _mSocket.Connect(server);
            }
            catch (Exception ex) {
                LastErrorCode = ErrorCode.ConnectionError;
                LastErrorString = ex.Message;
                return ErrorCode.ConnectionError;
            }

            try
            {
                byte[] bSend1 = { 3, 0, 0, 22, 17, 224, 0, 0, 0, 46, 0, 193, 2,
1, 0, 194, 2, 3, 0, 192, 1, 9 };

                switch (CPU) {
                    case CpuType.S7200:
                        //S7200: Chr(193) & Chr(2) & Chr(16) & Chr(0) 'Eigener
Tsap
                        bSend1[11] = 193;
```

```
                    bSend1[12] = 2;
                    bSend1[13] = 16;
                    bSend1[14] = 0;
                    //S7200: Chr(194) & Chr(2) & Chr(16) & Chr(0) 'Fremder
Tsap
                    bSend1[15] = 194;
                    bSend1[16] = 2;
                    bSend1[17] = 16;
                    bSend1[18] = 0;
                    break;
                case CpuType.S71200:
                case CpuType.S7300:
                    //S7300: Chr(193) & Chr(2) & Chr(1) & Chr(0)  'Eigener
Tsap
                    bSend1[11] = 193;
                    bSend1[12] = 2;
                    bSend1[13] = 1;
                    bSend1[14] = 0;
                    //S7300: Chr(194) & Chr(2) & Chr(3) & Chr(2)  'Fremder
Tsap
                    bSend1[15] = 194;
                    bSend1[16] = 2;
                    bSend1[17] = 3;
                    bSend1[18] = (byte)(Rack * 2 * 16 + Slot);
                    break;
                case CpuType.S7400:
                    //S7400: Chr(193) & Chr(2) & Chr(1) & Chr(0)  'Eigener
Tsap
                    bSend1[11] = 193;
                    bSend1[12] = 2;
                    bSend1[13] = 1;
                    bSend1[14] = 0;
                    //S7400: Chr(194) & Chr(2) & Chr(3) & Chr(3)  'Fremder
Tsap
                    bSend1[15] = 194;
                    bSend1[16] = 2;
                    bSend1[17] = 3;
                    bSend1[18] = (byte)(Rack * 2 * 16 + Slot);
                    break;
                case CpuType.S71500:
                    // Eigener Tsap
                    bSend1[11] = 193;
                    bSend1[12] = 2;
                    bSend1[13] = 0x10;
                    bSend1[14] = 0x2;
                    // Fredmer Tsap
                    bSend1[15] = 194;
                    bSend1[16] = 2;
                    bSend1[17] = 0x3;
                    bSend1[18] = (byte)(Rack * 2 * 16 + Slot);
                    break;
                default:
```

```
                    return ErrorCode.WrongCPU_Type;
                }

                _mSocket.Send(bSend1, 22, SocketFlags.None);
                if (_mSocket.Receive(bReceive, 22, SocketFlags.None) != 22)
                {
                    throw new
Exception(ErrorCode.WrongNumberReceivedBytes.ToString());
                }

                byte[] bsend2 = { 3, 0, 0, 25, 2, 240, 128, 50, 1, 0, 0, 255,
255, 0, 8, 0, 0, 240, 0, 0, 3, 0, 3, 1, 0 };
                _mSocket.Send(bsend2, 25, SocketFlags.None);

                if (_mSocket.Receive(bReceive, 27, SocketFlags.None) != 27)
                {
                    throw new
Exception(ErrorCode.WrongNumberReceivedBytes.ToString());
                }
            }
            catch(Exception exc)
            {
                LastErrorCode = ErrorCode.ConnectionError;
                LastErrorString = "Couldn't establish the connection to " + IP +
".\nMessage: " + exc.Message;
                return ErrorCode.ConnectionError;
            }

            return ErrorCode.NoError;
        }
```

ParseBytes() private method

Given a S7 variable type (Bool, Word, DWord, etc.), it converts the bytes in the appropriate C# format.

private **ParseBytes** (VarType varType, byte bytes, int varCount) : object

varType	VarType
bytes	byte
varCount	int
return	object

```
        private object ParseBytes(VarType varType, byte[] bytes, int varCount)
        {
            if (bytes == null) return null;

            switch (varType)
            {
                case VarType.Byte:
                    if (varCount == 1)
                        return bytes[0];
                    else
                        return bytes;
                case VarType.Word:
                    if (varCount == 1)
                        return Types.Word.FromByteArray(bytes);
                    else
                        return Types.Word.ToArray(bytes);
                case VarType.Int:
                    if (varCount == 1)
                        return Types.Int.FromByteArray(bytes);
                    else
                        return Types.Int.ToArray(bytes);
                case VarType.DWord:
                    if (varCount == 1)
                        return Types.DWord.FromByteArray(bytes);
                    else
                        return Types.DWord.ToArray(bytes);
                case VarType.DInt:
                    if (varCount == 1)
                        return Types.DInt.FromByteArray(bytes);
                    else
                        return Types.DInt.ToArray(bytes);
```

```
            case VarType.Real:
                if (varCount == 1)
                    return Types.Double.FromByteArray(bytes);
                else
                    return Types.Double.ToArray(bytes);
            case VarType.String:
                return Types.String.FromByteArray(bytes);
            case VarType.Timer:
                if (varCount == 1)
                    return Types.Timer.FromByteArray(bytes);
                else
                    return Types.Timer.ToArray(bytes);
            case VarType.Counter:
                if (varCount == 1)
                    return Types.Counter.FromByteArray(bytes);
                else
                    return Types.Counter.ToArray(bytes);
            case VarType.Bit:
                return null; //TODO
            default:
                return null;
        }
    }
```

Plc() public method

Creates a PLC object with all the parameters needed for connections. For S7-1200 and S7-1500, the default is rack = 0 and slot = 0. You need slot > 0 if you are connecting to external ethernet card (CP). For S7-300 and S7-400 the default is rack = 0 and slot = 2.

public **Plc** (CpuType cpu, string ip, Int16 slot) : System

cpu	CpuType	CpuType of the plc (select from the enum)
ip	string	Ip address of the plc
rack	System.Int16	slot of the CPU of the plc, usually it's 2 for S7300-S7400, 0 for S7-1200 and S7-1500. be set accordingly.
return	System	

```
public Plc(CpuType cpu, string ip, Int16 rack, Int16 slot)
{
    IP = ip;
    CPU = cpu;
    Rack = rack;
    Slot = slot;
}
```

S7.Net.Plc.Read C# (CSharp) Method

Plc Class Documentation | Usage Examples Of S7.Net.Plc::Read | Show file | Open project: killnine/s7netplus

Read() public method

Read and decode a certain number of bytes of the "VarType" provided. This can be used to read multiple consecutive variables of the same type (Word, DWord, Int, etc). If the read was not successful, check LastErrorCode or LastErrorString.

public **Read** (DataType dataType, int db, int startByteAdr, VarType varType, int varCount) : object

dataType	DataType	Data type of the memory area, can be DB, Timer, Counter, Merker(Memory), Input, Output
db	int	Address of the memory area (if you want to read DB1, this is set to 1). This must be set also
startByteAdr	int	Start byte address. If you want to read DB1.DBW200, this is 200.
varType	VarType	Type of the variable/s that you are reading
varCount	int	
return	object	

```
        public object Read(DataType dataType, int db, int startByteAdr, VarType
varType, int varCount)
        {
            int cntBytes = VarTypeToByteLength(varType, varCount);
            byte[] bytes = ReadBytes(dataType, db, startByteAdr, cntBytes);

            return ParseBytes(varType, bytes, varCount);
        }
```

Same methods

Plc::Read (string variable) : object

Usage Example

EXAMPLE #1

0

Show file

File: Main.cs **Project:** gilpark/qb_challenge

```
        static void Main(string[] args)
        {
            S7.Net.Plc plc     = new Plc(cpu: CpuType.S71200, ip: "10.1.10.142",
rack: 0, slot: 1);
            ErrorCode  errCode = plc.Open();

            var       b1    = (UInt16)plc.Read("MW100");
            ErrorCode write = plc.Write("MW102", 100);

            plc.Close();

            System.Console.WriteLine(b1);
            Console.ReadLine();
        }
```

ReadBytes() public method

Reads a number of bytes from a DB starting from a specified index. This handles more than 200 bytes with multiple requests. If the read was not successful, check LastErrorCode or LastErrorString.

public **ReadBytes** (DataType dataType, int db, int startByteAdr, int count) : byte[]

dataType	DataType	Data type of the memory area, can be DB, Timer, Counter, Merker(Memory), Input
db	int	Address of the memory area (if you want to read DB1, this is set to 1). This must be timers,etc.
startByteAdr	int	Start byte address. If you want to read DB1.DBW200, this is 200.
count	int	Byte count, if you want to read 120 bytes, set this to 120.
return	byte[]	

```
        public byte[] ReadBytes(DataType dataType, int db, int startByteAdr, int
count)
        {
            List<byte> resultBytes = new List<byte>();
            int index = startByteAdr;
            while (count > 0)
            {
                var maxToRead = (int)Math.Min(count, 200);
                byte[] bytes = ReadBytesWithASingleRequest(dataType, db, index,
maxToRead);
                if (bytes == null)
                    return resultBytes.ToArray();
                resultBytes.AddRange(bytes);
                count -= maxToRead;
                index += maxToRead;
            }
            return resultBytes.ToArray();
        }
```

ReadBytesWithASingleRequest() private method

private **ReadBytesWithASingleRequest** (DataType dataType, int db, int startByteAdr, int count) : byte[]

dataType	DataType
db	int
startByteAdr	int
count	int
return	byte[]

```
        private byte[] ReadBytesWithASingleRequest(DataType dataType, int db, int
startByteAdr, int count)
        {
            byte[] bytes = new byte[count];

            try
            {
                // first create the header
                int packageSize = 31;
                Types.ByteArray package = new ByteArray(packageSize);
                package.Add(ReadHeaderPackage());
                // package.Add(0x02);  // datenart
                package.Add(CreateReadDataRequestPackage(dataType, db,
startByteAdr, count));

                _mSocket.Send(package.array, package.array.Length,
SocketFlags.None);

                byte[] bReceive = new byte[512];
                int numReceived = _mSocket.Receive(bReceive, 512,
SocketFlags.None);
                if (bReceive[21] != 0xff)
                    throw new
Exception(ErrorCode.WrongNumberReceivedBytes.ToString());

                for (int cnt = 0; cnt < count; cnt++)
                    bytes[cnt] = bReceive[cnt + 25];

                return bytes;
            }
```

```
        catch (SocketException socketException)
        {
            LastErrorCode = ErrorCode.WriteData;
            LastErrorString = socketException.Message;
            return null;
        }
        catch (Exception exc)
        {
            LastErrorCode = ErrorCode.WriteData;
            LastErrorString = exc.Message;
            return null;
        }
    }
```

ReadClass() public method

Reads all the bytes needed to fill a class in C#, starting from a certain address, and set all the properties values to the value that are read from the plc. This reads ony properties, it doesn't read private variable or public variable without {get;set;} specified.

public **ReadClass** (object sourceClass, int db, int startByteAdr) : void

sourceClass	object	Instance of the class that will store the values
db	int	Index of the DB; es.: 1 is for DB1
startByteAdr	int	Start byte address. If you want to read DB1.DBW200, this is 200.
return	void	

```
        public void ReadClass(object sourceClass, int db, int startByteAdr = 0)
        {
            Type classType = sourceClass.GetType();
            int numBytes = Types.Class.GetClassSize(classType);
            // now read the package
            var resultBytes = ReadBytes(DataType.DataBlock, db, startByteAdr,
numBytes);
            // and decode it
            Types.Class.FromBytes(sourceClass, classType, resultBytes);
        }
```

ReadHeaderPackage() private method

Creates the header to read bytes from the plc

private **ReadHeaderPackage** (int amount = 1) : ByteArray

```
        private Types.ByteArray ReadHeaderPackage(int amount = 1)
        {
            //header size = 19 bytes
            var package = new Types.ByteArray(19);
            package.Add(new byte[] { 0x03, 0x00, 0x00 });
            //complete package size
            package.Add((byte)(19 + (12 * amount)));
            package.Add(new byte[] { 0x02, 0xf0, 0x80, 0x32, 0x01, 0x00, 0x00,
0x00, 0x00 });
            //data part size
            package.Add(Types.Word.ToByteArray((ushort)(2 + (amount * 12))));
            package.Add(new byte[] { 0x00, 0x00, 0x04 });
            //amount of requests
            package.Add((byte)amount);

            return package;
        }
```

ReadMultipleVars() public method

Reads multiple vars in a single request. You have to create and pass a list of DataItems and you obtain in response the same list with the values. Values are stored in the property "Value" of the dataItem and are already converted. If you don't want the conversion, just create a dataItem of bytes. DataItems must not be more than 20 (protocol restriction) and bytes must not be more than 200 + 22 of header (protocol restriction).

public **ReadMultipleVars** (List dataItems) : void

dataItems	List	List of dataitems that contains the list of variables that must be read. Maximum 20 data

return	void

```
        public void ReadMultipleVars(List<DataItem> dataItems)
        {
            int cntBytes = dataItems.Sum(dataItem =>
VarTypeToByteLength(dataItem.VarType, dataItem.Count));

            if (dataItems.Count > 20) throw new Exception("Too many vars
requested");
            if (cntBytes > 222) throw new Exception("Too many bytes requested");
//todo, proper TDU check + split in multiple requests

            try
            {
                // first create the header
                int packageSize = 19 + (dataItems.Count*12);
                Types.ByteArray package = new ByteArray(packageSize);
                package.Add(ReadHeaderPackage(dataItems.Count));
                // package.Add(0x02);  // datenart
                foreach (var dataItem in dataItems)
                {
                    package.Add(CreateReadDataRequestPackage(dataItem.DataType,
dataItem.DB, dataItem.StartByteAdr, VarTypeToByteLength(dataItem.VarType,
dataItem.Count)));
                }

                _mSocket.Send(package.array, package.array.Length,
SocketFlags.None);

                byte[] bReceive = new byte[512];
                int numReceived = _mSocket.Receive(bReceive, 512,
SocketFlags.None);
                if (bReceive[21] != 0xff) throw new
Exception(ErrorCode.WrongNumberReceivedBytes.ToString());

                int offset = 25;
                foreach (var dataItem in dataItems)
```

```
                {
                    int byteCnt = VarTypeToByteLength(dataItem.VarType,
dataItem.Count);
                    byte[] bytes = new byte[byteCnt];

                    for (int i = 0; i < byteCnt; i++)
                    {
                        bytes[i] = bReceive[i + offset];
                    }

                    offset += byteCnt + 4;

                    dataItem.Value = ParseBytes(dataItem.VarType, bytes,
dataItem.Count);
                }
            }
            catch (SocketException socketException)
            {
                LastErrorCode = ErrorCode.WriteData;
                LastErrorString = socketException.Message;
            }
            catch (Exception exc)
            {
                LastErrorCode = ErrorCode.WriteData;
                LastErrorString = exc.Message;
            }
        }
```

ReadStruct() public method

Reads all the bytes needed to fill a struct in C#, starting from a certain address, and return an object that can be casted to the struct.

public **ReadStruct** (Type		Type of the
db	int	Address of the DB.
startByteAdr	int	Start byte address. If you want to read DB1.DBW200, this is 200.
return	object	

```
        public object ReadStruct(Type structType, int db, int startByteAdr = 0)
        {
            int numBytes = Types.Struct.GetStructSize(structType);
            // now read the package
            var resultBytes = ReadBytes(DataType.DataBlock, db, startByteAdr,
numBytes);

            // and decode it
            return Types.Struct.FromBytes(structType, resultBytes);
        }
```

VarTypeToByteLength() private method

Given a S7 variable type (Bool, Word, DWord, etc.), it returns how many bytes to read.

private **VarTypeToByteLength** (VarType varType, int varCount = 1) : int

varType	VarType
varCount	int
return	int

```
private int VarTypeToByteLength(VarType varType, int varCount = 1)
{
    switch (varType)
    {
        case VarType.Bit:
            return varCount; //TODO
        case VarType.Byte:
            return (varCount < 1) ? 1 : varCount;
        case VarType.String:
            return varCount;
        case VarType.Word:
        case VarType.Timer:
        case VarType.Int:
        case VarType.Counter:
            return varCount * 2;
        case VarType.DWord:
        case VarType.DInt:
        case VarType.Real:
            return varCount * 4;
        default:
            return 0;
    }
}
```

Write() public method

Takes in input an object and tries to parse it to an array of values. This can be used to write many data, all of the same type. You must specify the memory area type, memory are address, byte start address and bytes count. If the read was not successful, check LastErrorCode or LastErrorString.

public **Write** (DataType dataType, int db, int startByteAdr, object value) : ErrorCode

dataType	DataType	Data type of the memory area, can be DB, Timer, Counter, Merker(Memory), Input, Output
db	int	Address of the memory area (if you want to read DB1, this is set to 1). This must be set also
startByteAdr	int	Start byte address. If you want to read DB1.DBW200, this is 200.
value	object	Bytes to write. The lenght of this parameter can't be higher than 200. If you need more, use
return	ErrorCode	

```
        public ErrorCode Write(DataType dataType, int db, int startByteAdr,
object value)
        {
            byte[] package = null;

            switch (value.GetType().Name)
            {
                case "Byte":
                    package = Types.Byte.ToByteArray((byte)value);
                    break;
                case "Int16":
                    package = Types.Int.ToByteArray((Int16)value);
                    break;
                case "UInt16":
                    package = Types.Word.ToByteArray((UInt16)value);
                    break;
                case "Int32":
                    package = Types.DInt.ToByteArray((Int32)value);
                    break;
                case "UInt32":
                    package = Types.DWord.ToByteArray((UInt32)value);
                    break;
                case "Double":
                    package = Types.Double.ToByteArray((Double)value);
                    break;
                case "Byte[]":
```

```
                package = (byte[])value;
                break;
            case "Int16[]":
                package = Types.Int.ToByteArray((Int16[])value);
                break;
            case "UInt16[]":
                package = Types.Word.ToByteArray((UInt16[])value);
                break;
            case "Int32[]":
                package = Types.DInt.ToByteArray((Int32[])value);
                break;
            case "UInt32[]":
                package = Types.DWord.ToByteArray((UInt32[])value);
                break;
            case "Double[]":
                package = Types.Double.ToByteArray((double[])value);
                break;
            case "String":
                package = Types.String.ToByteArray(value as string);
                break;
            default:
                return ErrorCode.WrongVarFormat;
        }
        return WriteBytes(dataType, db, startByteAdr, package);
    }
```

Same methods

Plc::Write (string variable, object value) : ErrorCode

Usage Example

EXAMPLE #1

0

Show file

File: Main.cs **Project:** gilpark/qb_challenge

```
        static void Main(string[] args)
        {
            S7.Net.Plc plc     = new Plc(cpu: CpuType.S71200, ip: "10.1.10.142",
rack: 0, slot: 1);
            ErrorCode  errCode = plc.Open();

            var       b1    = (UInt16)plc.Read("MW100");
```

```
    ErrorCode write = plc.Write("MW102", 100);

    plc.Close();

    System.Console.WriteLine(b1);
    Console.ReadLine();
}
```

WriteBytes() public method

Write a number of bytes from a DB starting from a specified index. This handles more than 200 bytes with multiple requests. If the write was not successful, check LastErrorCode or LastErrorString.

public **WriteBytes** (DataType dataType, int db, int startByteAdr, byte value) : ErrorCode

dataType	DataType	Data type of the memory area, can be DB, Timer, Counter, Merker(Memory), Inpu
db	int	Address of the memory area (if you want to read DB1, this is set to 1). This must b timers,etc.
startByteAdr	int	Start byte address. If you want to write DB1.DBW200, this is 200.
value	byte	Bytes to write. If more than 200, multiple requests will be made.
return	ErrorCode	

```
        public ErrorCode WriteBytes(DataType dataType, int db, int startByteAdr,
byte[] value)
        {
            int localIndex = 0;
            int count = value.Length;
            while (count > 0)
            {
                var maxToWrite = (int)Math.Min(count, 200);
                ErrorCode lastError = WriteBytesWithASingleRequest(dataType, db,
startByteAdr + localIndex, value.Skip(localIndex).Take(maxToWrite).ToArray());
                if (lastError != ErrorCode.NoError)
                {
                    return lastError;
                }
                count -= maxToWrite;
                localIndex += maxToWrite;
            }
            return ErrorCode.NoError;
        }
```

WriteBytesWithASingleRequest() private method

Writes up to 200 bytes to the plc and returns NoError if successful. You must specify the memory area type, memory are address, byte start address and bytes count. If the write was not successful, check LastErrorCode or LastErrorString.

private **WriteBytesWithASingleRequest** (DataType dataType, int db, int startByteAdr, byte value) : ErrorCo

dataType	DataType	Data type of the memory area, can be DB, Timer, Counter, Merker(Memory), Inpu
db	int	Address of the memory area (if you want to read DB1, this is set to 1). This must b timers,etc.
startByteAdr	int	Start byte address. If you want to read DB1.DBW200, this is 200.
value	byte	Bytes to write. The lenght of this parameter can't be higher than 200. If you need m
return	ErrorCode	

```
        private ErrorCode WriteBytesWithASingleRequest(DataType dataType, int db,
int startByteAdr, byte[] value)
        {
            byte[] bReceive = new byte[513];
            int varCount = 0;

            try
            {
                varCount = value.Length;
                // first create the header
                int packageSize = 35 + value.Length;
                Types.ByteArray package = new Types.ByteArray(packageSize);

                package.Add(new byte[] { 3, 0, 0 });
                package.Add((byte)packageSize);
                package.Add(new byte[] { 2, 0xf0, 0x80, 0x32, 1, 0, 0 });
                package.Add(Types.Word.ToByteArray((ushort)(varCount - 1)));
                package.Add(new byte[] { 0, 0x0e });
                package.Add(Types.Word.ToByteArray((ushort)(varCount + 4)));
                package.Add(new byte[] { 0x05, 0x01, 0x12, 0x0a, 0x10, 0x02 });
                package.Add(Types.Word.ToByteArray((ushort)varCount));
                package.Add(Types.Word.ToByteArray((ushort)(db)));
                package.Add((byte)dataType);
                var overflow = (int)(startByteAdr * 8 / 0xffffU); // handles
words with address bigger than 8191
```

```
                package.Add((byte)overflow);
                package.Add(Types.Word.ToByteArray((ushort)(startByteAdr * 8)));
                package.Add(new byte[] { 0, 4 });
                package.Add(Types.Word.ToByteArray((ushort)(varCount * 8)));

                // now join the header and the data
                package.Add(value);

                _mSocket.Send(package.array, package.array.Length,
SocketFlags.None);

                int numReceived = _mSocket.Receive(bReceive, 512,
SocketFlags.None);
                if (bReceive[21] != 0xff)
                {
                    throw new
Exception(ErrorCode.WrongNumberReceivedBytes.ToString());
                }

                return ErrorCode.NoError;
            }
            catch (Exception exc)
            {
                LastErrorCode = ErrorCode.WriteData;
                LastErrorString = exc.Message;
                return LastErrorCode;
            }
        }
```

WriteClass() public method

Writes a C# class to a DB in the plc

public **WriteClass** (object classValue, int db, int startByteAdr) : ErrorCode

classValue	object	The class
db	int	Db addres
startByteAdr	int	Start byte
return	ErrorCode	

```
        public ErrorCode WriteClass(object classValue, int db, int startByteAdr =
0)
        {
            var bytes = Types.Class.ToBytes(classValue).ToList();
            var errCode = WriteBytes(DataType.DataBlock, db, startByteAdr,
bytes.ToArray());
            return errCode;
        }
```

WriteStruct() public method

Writes a C# struct to a DB in the plc

public **WriteStruct** (object structValue, int db, int startByteAdr) : ErrorCode

structValue	object	The struct
db	int	Db address
startByteAdr	int	Start bytes
return	ErrorCode	

```
        public ErrorCode WriteStruct(object structValue, int db, int startByteAdr
= 0)
        {
            var bytes = Types.Struct.ToBytes(structValue).ToList();
            var errCode = WriteBytes(DataType.DataBlock ,db, startByteAdr,
bytes.ToArray());
            return errCode;
        }
```