# Home

Edit New Page

Jump to bottom

FalcoGoodbody edited this page on Feb 9 · 35 revisions

# S7.Net documentation

## How to download s7.Net

The official repository is on GitHub (https://github.com/killnine/s7netplus), you can also download the library directly from NuGet (https://www.nuget.org/packages/S7netplus/).

## What is S7.Net

S7.Net is a plc driver that works only with Siemens PLC and only with Ethernet connection. This means that your plc must have a Profinet CPU or a profinet external card (CPxxx card). S7.Net is written entirely in C#, so you can debug it easily without having to go through native dlls.

## Supported PLC

S7.Net is compatible with S7-200, S7-300, S7-400, S7-1200, S7-1500.

## Getting started with S7.Net

To get started with S7.Net you have to download and include the S7.Net.dll in your project. You can do this by downloading the NuGet package, or by downloading the sources and compile them.

## Create a PLC instance, connect and disconnect

To create an instance of the driver you need to use this constructor:

```
public Plc(CpuType cpu, string ip, Int16 rack, Int16 slot)
```

- **cpu**: this specify what CPU you are connecting to. The supported CPU are:

```
public enum CpuType {
    S7200 = 0,
    S7300 = 10,
    S7400 = 20,
    S71200 = 30,
    S71500 = 40,
}
```

- **ip**: specify the IP address of the CPU or of the external Ethernet card
- **rack**: this contains the rack of the plc, that you can find in hardware configuration in Step7
- **slot**: this is the slot of the CPU, that you can find in hardware configuration in Step7

Example:

This code creates a Plc object for a S7-300 plc at the IP address 127.0.0.1, for a plc in rack 0 with the cpu in slot 2:

```
Plc plc = new Plc(CpuType.S7300, "127.0.0.1", 0, 2);
```

## Connecting to the PLC

```
public void Open()
```

For example this line of code open the connection:

```
plc.Open();
```

## Disconnecting from the PLC

```
public void Close()
```

For example this closes the connection:

```
plc.Close();
```

## Error handling

Any method can cause a `PlcException` on various errors. You should implement a proper error handling. The `PlcException` provides an `ErrorCode` and an adequate error message.

These are the types of errors:

```
public enum ErrorCode
{
    NoError = 0,
    WrongCPU_Type = 1,
    ConnectionError = 2,
    IPAddressNotAvailable,
    WrongVarFormat = 10,
    WrongNumberReceivedBytes = 11,
    SendData = 20,
    ReadData = 30,
    WriteData = 50
}
```

## Check PLC availability

To check if the plc is available (opens a Socket) you can use the property

```
public bool IsAvailable
```

When you check this property, the driver will try to connect to the plc and returns true if it can connect, false otherwise.

## Check PLC connection

Checking the plc connection is trivial, because you have to check if the PC socket is connected but also if the PLC is still connected at the other side of the socket. The property that you have to check in this case is:

```
public bool IsConnected
```

This property can be checked after you called the method Open() and the result was a success, to check if the connection is still alive.

## Read bytes / Write bytes

The library offers several methods to read variables. The basic one and the most used is ReadBytes.

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

public void WriteBytes(DataType dataType, int db, int startByteAdr, byte[] value)
```

This reads all the bytes you specify from a given memory location. This method handles multiple requests automatically in case the number of bytes exceeds the maximum bytes that can be transferred in a single request.

- **dataType**: you have to specify the memory location with the enum DataType

```
public enum DataType
{
    Input = 129,
    Output = 130,
    Memory = 131,
    DataBlock = 132,
    Timer = 29,
    Counter = 28
}
```

- **db**: the address of the dataType, for example if you want to read DB1, this field is "1"; if you want to read T45, this field is 45.
- **startByteAdr**: the address of the first byte that you want to read, for example if you want to read DB1.DBW200, this is 200.
- **count**: contains how many bytes you want to read.
- **value[ ]**: array of bytes to be written to the plc.

Example: This method reads the first 200 bytes of DB1:

```
var bytes = plc.ReadBytes(DataType.DataBlock, 1, 0, 200);
```

## Read and decode / Write decoded

This method permits to read and receive an already decoded result based on the varType provided. This is useful if you read several fields of the same type (for example 20 consecutive DBW). If you specify VarType.Byte, it has the same functionality as ReadBytes.

```
public object Read(DataType dataType, int db, int startByteAdr, VarType varType, int varCo

public void Write(DataType dataType, int db, int startByteAdr, object value)
```

- **dataType**: you have to specify the memory location with the enum DataType

```
public enum DataType
{
    Input = 129,
    Output = 130,
    Memory = 131,
    DataBlock = 132,
    Timer = 29,
    Counter = 28
}
```

- **db**: the address of the dataType, for example if you want to read DB1, this field is "1"; if you want to read T45, this field is 45.
- **startByteAdr**: the address of the first byte that you want to read, for example if you want to read DB1.DBW200, this is 200.
- **varType**: specify the data that you want to get your bytes converted.

```
public enum VarType
{
    Bit,
    Byte,
    Word,
    DWord,
    Int,
    DInt,
    Real,
    String,
    StringEx,
    Timer,
    Counter
}
```

- **count**: contains how many variables you want to read.
- **value**: array of values to be written to the plc. It can be a single value or an array, just remember that the type is unique (for example array of double, array of int, array of shorts, etc..)

Example: This method reads the first 20 DWords of DB1:

```
var dwords = plc.Read(DataType.DataBlock, 1, 0, VarType.DWord, 20);
```

## Read a single variable / Write a single variable

This method reads a single variable from the plc, by parsing the string and returning the correct result. While this is the easiest method to get started, is very inefficient because the driver sends a TCP request for every variable.

```
public object Read(string variable)

public void Write(string variable, object value)
```

- **variable**: specify the variable to read by using strings like “DB1.DBW20”, “T45”, “C21”, “DB1.DBD400”, etc.

Example: This reads the variable DB1.DBW0. The result must be cast to ushort to get the correct 16-bit format in C#.

```
ushort result = (ushort)plc.Read("DB1.DBW0");
```

## Read a struct / Write a struct

This method reads all the bytes from a specified DB needed to fill a struct in C#, and returns the struct that contains the values. It is recommended to use when you want to read many variables in a single data block in some continuous memory range.

The "read struct" and "write struct" methods do not support strings.

```
public object ReadStruct(Type structType, int db, int startByteAdr = 0)

public void WriteStruct(object structValue, int db, int startByteAdr = 0)
```

- **structType**: Type of the struct to be read, for example: typeOf(MyStruct))
- **db**: index of the DB to read
- **startByteAdr**: specified the first address of the byte to read (the default is zero).

Example: Define a DataBlock in the plc like:

| Adresse | Name | Typ | Anfangswert |
|---|---|---|---|
| 0.0 | | STRUCT | |
| +0.0 | varBool0 | BOOL | FALSE |
| +0.1 | varBool1 | BOOL | FALSE |
| +0.2 | varBool2 | BOOL | FALSE |
| +0.3 | varBool3 | BOOL | FALSE |
| +0.4 | varBool4 | BOOL | FALSE |
| +0.5 | varBool5 | BOOL | FALSE |
| +0.6 | varBool6 | BOOL | FALSE |
| +1.0 | varByte0 | BYTE | B#16#0 |
| +2.0 | varByte1 | BYTE | B#16#0 |
| +4.0 | varWord | WORD | W#16#0 |
| +6.0 | varReal | REAL | 1.230000e+000 |
| +10.0 | varBool7 | BOOL | TRUE |
| +12.0 | varReal1 | REAL | 8.506780e+002 |
| +16.0 | varbyte2 | BYTE | B#16#55 |
| +18.0 | varDWord | DWORD | DW#16#1234567 |
| =22.0 | | END_STRUCT | |

Then add a struct into your .Net application that is similiar to the DB in the plc:

```
public struct testStruct
{
    public bool varBool0;
    public bool varBool1;
    public bool varBool2;
    public bool varBool3;
    public bool varBool4;
    public bool varBool5;
    public bool varBool6;
    public byte varByte0;
    public byte varByte1;
    public ushort varWord0;
    public float varReal0;
    public bool varBool7;
    public float varReal1;
    public byte varByte2;
    public UInt32 varDWord;
}
```

then add the code to read or write the complete struct

```
// reads a struct from DataBlock 1 at StartAddress 0
testStruct myTestStruct = (testStruct) plc.ReadStruct(typeof(testStruct), 1, 0);
```

## Read a class / Write a class

This method reads all the bytes from a specified DB needed to fill a class in C#. The class is passed as reference and values are assigned by using reflection. It is recommended to use when you want to read many variables in a single data block in some continuous memory range.

The "read class" and "write class" methods do not support strings.

```
public void ReadClass(object sourceClass, int db, int startByteAdr = 0)

public void WriteClass(object classValue, int db, int startByteAdr = 0)
```

- **sourceClass**: instance of the class that you want to assign the values
- **db**: index of the DB to read
- **startByteAdr**: specified the first address of the byte to read (the default is zero). Example: Define a DataBlock in the plc like:

| Adresse | Name | Typ | Anfangswert |
|---|---|---|---|
| 0.0 | | STRUCT | |
| +0.0 | varBool0 | BOOL | FALSE |
| +0.1 | varBool1 | BOOL | FALSE |
| +0.2 | varBool2 | BOOL | FALSE |
| +0.3 | varBool3 | BOOL | FALSE |
| +0.4 | varBool4 | BOOL | FALSE |
| +0.5 | varBool5 | BOOL | FALSE |
| +0.6 | varBool6 | BOOL | FALSE |
| +1.0 | varByte0 | BYTE | B#16#0 |
| +2.0 | varByte1 | BYTE | B#16#0 |
| +4.0 | varWord | WORD | W#16#0 |
| +6.0 | varReal | REAL | 1.230000e+000 |
| +10.0 | varBool7 | BOOL | TRUE |
| +12.0 | varReal1 | REAL | 8.506780e+002 |
| +16.0 | varbyte2 | BYTE | B#16#55 |
| +18.0 | varDWord | DWORD | DW#16#1234567 |
| =22.0 | | END_STRUCT | |

Then add a struct into your .Net application that is similiar to the DB in the plc:

```
public class TestClass
{
    public bool varBool0 { get; set;}
    public bool varBool1 { get; set;}
    public bool varBool2 { get; set;}
    public bool varBool3 { get; set;}
    public bool varBool4 { get; set;}
    public bool varBool5 { get; set;}
    public bool varBool6 { get; set;}

    public byte varByte0 { get; set;}
    public byte varByte1 { get; set;}

    public ushort varWord0 { get; set;}
```

```
    public float varReal0 { get; set;}
    public bool varBool7 { get; set;}
    public float varReal1 { get; set;}

    public byte varByte2 { get; set;}
    public UInt32 varDWord { get; set;}
}
```

then add the code to read or write the complete class

```
// reads a class from DataBlock 1, startAddress 0
TestClass myTestClass = new TestClass();
plc.ReadClass(myTestClass, 1, 0);
```

## Read multiple variables

This method reads multiple variables in a single request. The variables can be locatet in the same or in different data blocks.

```
public void Plc.ReadMultibleVars(List<DataItem> dataItems);
```

- **List**<>: you have to specifiy a list of DataItem which contains all data items you want to read

Example: this method reads several variables from a data block

define the data items first

```
private static DataItem varBit = new DataItem()
{
    DataType = DataType.DataBlock,
    VarType = VarType.Bit,
    DB = 83,
    BitAdr = 0,
    Count = 1,
    StartByteAdr = 0,
    Value = new object()
};

private static DataItem varByteArray = new DataItem()
{
    DataType = DataType.DataBlock,
    VarType = VarType.Byte,
    DB = 83,
    BitAdr = 0,
    Count = 100,
    StartByteAdr = 0,
    Value = new object()
```

```
};

private static DataItem varInt = new DataItem()
{
    DataType = DataType.DataBlock,
    VarType = VarType.Int,
    DB = 83,
    BitAdr = 0,
    Count = 1,
    StartByteAdr = 102,
    Value = new object()
};

private static DataItem varReal = new DataItem()
{
    DataType = DataType.DataBlock,
    VarType = VarType.Real,
    DB = 83,
    BitAdr = 0,
    Count = 1,
    StartByteAdr = 112,
    Value = new object()
};

private static DataItem varString = new DataItem()
{
    DataType = DataType.DataBlock,
    VarType = VarType.StringEx,
    DB = 83,
    BitAdr = 0,
    Count = 20,         // max lengt of string
    StartByteAdr = 116,
    Value = new object()
};

private static DataItem varDateTime = new DataItem()
{
    DataType = DataType.DataBlock,
    VarType = VarType.DateTime,
    DB = 83,
    BitAdr = 0,
    Count = 1,
    StartByteAdr = 138,
    Value = new object()
};
```

Then define a list where the DataItems will be stored in

```
private static List<DataItem> dataItemsRead = new List<DataItem>();
```

add the data items to the list

```
dataItemsRead.Add(varBit);
dataItemsRead.Add(varByteArray);
dataItemsRead.Add(varInt);
dataItemsRead.Add(varReal);
dataItemsRead.Add(varString);
dataItemsRead.Add(varDateTime);
```

open the connection to the plc and read the items at once

```
myPLC.Open();

// read the list of variables
myPLC.ReadMultipleVars(dataItemsRead);

// close the connection
myPLC.Close();

// access the values of the list
Console.WriteLine("Int:" + dataItemsRead[2].Value);
```

## Write multiple variables

This method writes multiple variables in a single request.

```
public void Plc.Write(Array[DataItem] dataItems);
```

- **Array[]** you have to specifiy an array of DataItem which contains all the items you want to write

Example: this method writes multiple variables into one data block

define the data items

```
 private static DataItem varWordWrite = new DataItem()
{
    DataType = DataType.DataBlock,
    VarType = VarType.Word,
    DB = 83,
    BitAdr = 0,
    Count = 1,
    StartByteAdr = 146,
    Value = new object()
};

private static DataItem varIntWrite = new DataItem()
{
    DataType = DataType.DataBlock,
```

```
        VarType = VarType.Int,
        DB = 83,
        BitAdr = 0,
        Count = 1,
        StartByteAdr = 148,
        Value = new object()
    };

    private static DataItem varDWordWrite = new DataItem()
    {
        DataType = DataType.DataBlock,
        VarType = VarType.DWord,
        DB = 83,
        BitAdr = 0,
        Count = 1,
        StartByteAdr = 150,
        Value = new object()
    };

    private static DataItem varDIntWrite = new DataItem()
    {
        DataType = DataType.DataBlock,
        VarType = VarType.DInt,
        DB = 83,
        BitAdr = 0,
        Count = 1,
        StartByteAdr = 154,
        Value = new object()
    };

    private static DataItem varRealWrite = new DataItem()
    {
        DataType = DataType.DataBlock,
        VarType = VarType.Real,
        DB = 83,
        BitAdr = 0,
        Count = 1,
        StartByteAdr = 158,
        Value = new object()
    };

    private static DataItem varStringWrite = new DataItem()
    {
        DataType = DataType.DataBlock,
        VarType = VarType.StringEx,
        DB = 83,
        BitAdr = 0,
        Count = 20,
        StartByteAdr = 162,
        Value = new object()
    };
```

Asign the values to the data items. Be aware to use the correct data conversion for the variables to fit into the S7-data types

```
// asign values to the variable to be written
varWordWrite.Value = (ushort)67;
varIntWrite.Value = (ushort)33;
varDWordWrite.Value = (uint)444;
varDIntWrite.Value = 6666;
varRealWrite.Value = 77.89;
varStringWrite.Value = "Writting";
```

Then define a list to store the data items and add the created items to the list

```
private static List<DataItem> dataItemsWrite = new List<DataItem>();

// add data items to list of data items to write
dataItemsWrite.Add(varWordWrite);
dataItemsWrite.Add(varIntWrite);
dataItemsWrite.Add(varDWordWrite);
dataItemsWrite.Add(varDIntWrite);
dataItemsWrite.Add(varRealWrite);
dataItemsWrite.Add(varStringWrite);
```

Finally open a connection to the plc and write the items at once. Use `.ToArray()` to convert the list into an array.

```
// open the connection
myPLC.Open();

// write the items
myPLC.Write(dataItemsWrite.ToArray());

// close the connection
myPLC.Close();
```

Value conversion

+ Add a custom sidebar

### Clone this wiki locally

```
https://github.com/S7NetPlus/s7netplus.wiki.git
```