Project overview

Sharp7 is the C# port of Snap7 Client. It’s not a wrapper, i.e. you don’t have an interface code that loads snap7.dll (or .so) but it’s a pure C# implementation of the S7Protocol.

Sharp7 is deployed as a single source file that contains some classes that you can use directly in your .NET project to communicate with S7 PLCs.

It’s designed to work with small hardware .NET-based or even for large projects which don’t needs of extended control functions.

Main features

- Fully standard “safe managed” C# code without any dependencies.
- Virtually every hardware with an Ethernet adapter able to run a .NET Core can be connected to an S7 PLC.
- Packed protocol headers to improve performances.
- Helper class to access to all S7 types without worrying about Little-Big endian conversion.
- Compatible with Universal Windows Platform including Win10 IoT for Raspberry.
- One single file.
- Compatible with Snap7.net.cs, plug and play replaceable.
- No additional libraries to deploy.

Licensing

Sharp7 is distributed as a source code library under **GNU Library** or **Lesser General Public License version 3.0 (LGPLv3)**.

Basically this means that you can distribute your commercial firmware containing Sharp7 without the requirement to distribute the source code of your application and without the requirement that your firmware be itself distributed under LGPL. A small mention is however appreciated if you include it in your applications.

Disclaimer of Warranty

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE

Sharp7 deploy

Sharp7 is a single file classes library.

For C# users : just add it in your C# project.

For VB users refer to the example, you need to create the assembly Sharp7.net

S7Client reference

Administrative functions

These methods allow controlling the behavior a Client Object.

Function	Purpose
ConnectTo	Connects a Client Object to a PLC.
SetConnectionType	Sets the connection type (PG/OP/S7Basic)
SetConnectionParams	Sets Address, Local and Remote TSAP for the connection.
Connect	Connects a Client Object to a PLC with implicit parameters.
Disconnect	Disconnects a Client.

SetConnectionType

Description

Sets the connection resource type, i.e. the way in which the Clients connects to a PLC.

Declaration

```
public void SetConnectionType(short ConnectionType)
```

Parameters

	Type	Dir.	
ConnectionType	short	In	See the table

Connection type table

Connection Type	Value	Helper Const
PG	0x01	S7.PG
OP	0x02	S7.OP
S7 Basic	0x03..0x10	S7.S7_BASIC

ConnectTo

Description

Connects the client to the hardware at (IP, Rack, Slot) Coordinates.

Declaration

```
public int ConnectTo(String Address, int Rack, int Slot)
```

Parameters

	Type	Dir.	
Address	String	In	PLC/Equipment IPV4 Address ex. “192.168.1.12”
Rack	int	In	PLC Rack number (see below)
Slot	int	In	PLC Slot number (see below)

Return value

- 0 : The Client is successfully connected (or was already connected).
- Other values : see the Errors Code List.

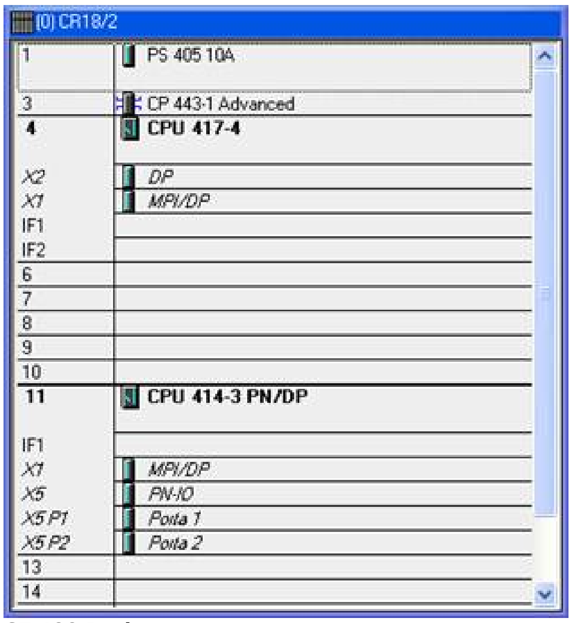
Rack and Slot

In addition to the IP Address, that we all understand, there are two other parameters that index the unit : **Rack** (0..7) and **Slot** (1..31) that you find into the hardware configuration of your project, for a physical component, or into the Station Configuration manager for WinAC.

There is however some special cases for which those values are fixed or can work with a default as you can see in the next table.

	Rack	Slot	
S7 300 CPU	0	2	Always
S7 400 CPU	Not fixed		Follow the hardware configuration.
WinAC CPU	Not fixed		Follow the hardware configuration.
S7 1200 CPU	0	0	Or 0, 1
S7 1500 CPU	0	0	Or 0, 1
WinAC IE	0	0	Or follow Hardware configuration.

Let’s see some examples of hardware configuration:



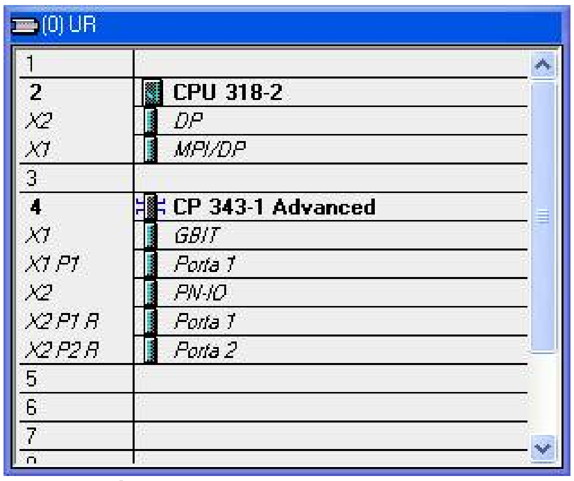
(0) CR18/2

1	PS 405 10A
3	CP 443-1 Advanced
4	CPU 417-4
X2	DP
X1	MPI/DP
IF1	
IF2	
6	
7	
8	
9	
10	
11	CPU 414-3 PN/DP
IF1	
X1	MPI/DP
X5	PN-IO
X5 P1	Porta 1
X5 P2	Porta 2
13	
14	

S7 400 Rack

	Rack	Slot
CPU 1	0	4
CPU 2	0	11

The same concept for WinAC CPU which index can vary inside the PC Station Rack.



(0) UR

1	
2	CPU 318-2
X2	DP
X1	MPI/DP
3	
4	CP 343-1 Advanced
X1	GBIT
X1 P1	Porta 1
X2	PN-IO
X2 P1 R	Porta 1
X2 P2 R	Porta 2
5	
6	
7	

S7300 Rack

	Rack	Slot
CPU	0	2

S7300 CPU is always present in Rack 0 at Slot 2

SetConnectionParams

Description

Sets internally (IP, LocalTSAP, RemoteTSAP) Coordinates.

Declaration

```
public void SetConnectionParams(String Address, int LocalTSAP,
    int RemoteTSAP)
```

Parameters

	Type	Dir.	
Address	String	In	PLC/Equipment IPV4 Address ex. “192.168.1.12”
LocalTSAP	int	In	Local TSAP (PC TSAP)
RemoteTSAP	int	In	Remote TSAP (PLC TSAP)

Remarks

This function must be called just before **Connect().**

Connect

Description

Connects the client to the PLC with the parameters specified in the previous call of **ConnectTo()** or **SetConnectionParams().**

Declaration

```
public int Connect()
```

Return value

- 0 : The Client is successfully connected (or was already connected).
- Other values : see the Errors Code List.

Remarks

This function can be called only after a previous of `ConnectTo()` or `SetConnectionParams()` which internally sets Address and TSAPs.

Disconnect

Description

Disconnects “gracefully” the Client from the PLC.

Declaration

```
public void Disconnect()
```

Remarks

This function can be called safely multiple times.
After calling this function LastError = 0 and Connected = false.

Base Data I/O functions

These functions allow the Client to exchange data with a PLC.

Function	Purpose
ReadArea	Reads a data area from a PLC.
WriteArea	Writes a data area into a PLC.
ReadMultiVars	Reads different kind of variables from a PLC simultaneously.
WriteMultiVars	Writes different kind of variables into a PLC simultaneously.

ReadArea

Description

This is the main function to read data from a PLC.
With it you can read DB, Inputs, Outputs, Merkers, Timers and Counters.

Declaration

```
public int ReadArea(int Area, int DBNumber, int Start, int Amount, int WordLen, byte[]
Buffer)
```

```
public int ReadArea(int Area, int DBNumber, int Start, int Amount, int  WordLen, byte[]
Buffer, ref int BytesRead)
```

Parameters

	Type	Dir.	Mean
Area	int	In	Area identifier.
DBNumber	int	In	DB Number if Area = S7AreaDB, otherwise is ignored.
Start	int	In	Offset to start
Amount	int	In	Amount of elements to read (1)
Wordlen	int	In	Word size (2)
Buffer	Byte Buffer	In	Buffer
BytesRead	int	Out	Number of bytes read (3)

(1) Note the use of the parameter name "amount", it means quantity of elements, not byte size.

Area table

	Value	Mean
S7Consts.S7AreaPE	0x81	Process Inputs.
S7Consts.S7AreaPA	0x82	Process Outputs.
S7Consts.S7AreaMK	0x83	Merkers.
S7Consts.S7AreaDB	0x84	DB
S7Consts.S7AreaCT	0x1C	Counters.
S7Consts.S7AreaTM	0x1D	Timers

WordLen table

	Value	Mean
S7WLBit	0x01	Bit (inside a word)
S7WLByte	0x02	Byte (8 bit)
S7WLWord	0x04	Word (16 bit)
S7WLDWord	0x06	Double Word (32 bit)
S7WLReal	0x08	Real (32 bit float)
S7WLCounter	0x1C	Counter (16 bit)
S7WLTimer	0x1D	Timer (16 bit)

Return value

- 0 : The function was accomplished with no errors.
- Other values : see the Errors Code List.

Remarks

As said, every data packet exchanged with a PLC must fit in a PDU, whose size is fixed and varies from 240 up to 960 bytes.

This function completely hides this concept, the data that you can transfer in a single call depends only on the size available of the data area (i.e. obviously, you cannot read 1024 bytes from a DB whose size is 300 bytes).

If this data size exceeds the PDU size, the packet is automatically split across more subsequent transfers.

If either **S7AreaCT** or **S7AreaTM** is selected, WordLen must be either **S7WLCounter** or **S7WLTimer** (However no error is raised and the values are internally fixed).

Your buffer should be large enough to receive the data.

Particularly:

Buffer size (byte) = Word size * Amount

Where:

	Word size
S7Consts.S7WLBit	1
S7Consts.S7WLByte	1
S7Consts.S7WLWord	2
S7Consts.S7WLDWord	4
S7Consts.S7WLReal	4
S7Consts.S7WLCounter	2
S7Consts.S7WLTimer	2

Notes

(2) When **WordLen**=S7WLBit the Offset (Start) must be expressed in bits.
Ex. The Start for DB4.DBX 10.3 is (10*8)+3 = **83**.

(3) Since Amount is the number of elements read, BytesRead returns the effective number of **bytes**.

WriteArea

Description

This is the main function to write data into a PLC. It's the complementary function of ReadArea(), the parameters and their meanings are the same.

The only difference is that the data is transferred from the byte buffer into PLC.

Declaration

```
public int WriteArea(int Area, int DBNumber, int Start, int Amount, int WordLen, byte[]
Buffer)
```

```
public int WriteArea(int Area, int DBNumber, int Start, int Amount, int WordLen, byte[]
Buffer, ref int BytesWritten)
```

See **ReadArea()** for parameters and remarks.

Use S7 helper methods to insert S7 data types (int, word, real …) into the byte buffer.

ReadMultiVars

Description

This is function allows to read different kind of variables from a PLC in a single call.
With it you can read DB, Inputs, Outputs, Merkers, Timers and Counters.

Declaration

```
public int ReadMultiVars(S7DataItem[] Items, int ItemsCount)
```

Parameters

	Type	Dir.	
Item	S7DataItem[]	In	See below
ItemsCount	integer	In	Number of Items to read.

S7DataItem struct

	Type	Dir.	
Area	int	In	Area identifier.
Wordlen	int	In	Word size
Result	int	Out	Item operation result (2)
DBNumber	int	In	DB Number if Area = S7AreaDB, otherwise is ignored.
Start	int	In	Offset to start
Amount	int	In	Amount of words to read (1)
pData	IntPtr	In	Pointer to user Buffer

(1) Note the use of the parameter name "amount", it means quantity of words, not byte size.

Return value

- 0 : The function was accomplished with no errors.
- Other values : see the Errors Code List.

(2) Since could happen that some variables are read, some other not because maybe they don't exist in PLC. **Is important to check the single item Result**.

Remarks

To use ReadMultiVars a special class is supplied **S7MultiVar** that avoids the use of unsafe code (since there is a IntPtr)

Due the different kind of variables involved , there is no split feature available for this function, so **the maximum data size must not exceed the PDU size**.

The advantage of this function becomes big when you have many small noncontiguous variables to be read.

WriteMultiVars

Description

This is function allows to write different kind of variables from a PLC in a single call.
With it you can read DB, Inputs, Outputs, Merkers, Timers and Counters.

Declaration

```
public int WriteMultiVars(S7DataItem[] Items, int ItemsCount)
```

Parameters

	Type	Dir.	
Item	S7DataItem[]	In	See below
ItemsCount	integer	In	Number of Items to write.

S7DataItem struct

	Type	Dir.	
Area	int	In	Area identifier.
Wordlen	int	In	Word size
Result	int	Out	Item operation result (2)
DBNumber	int	In	DB Number if Area = S7AreaDB, otherwise is ignored.
Start	int	In	Offset to start
Amount	int	In	Amount of words to write (1)
pData	IntPtr	In	Pointer to user Buffer

(2) Note the use of the parameter name "amount", it means quantity of words, not byte size.

Return value

- 0 : The function was accomplished with no errors.
- Other values : see the Errors Code List.

(2) Since could happen that some variables are written, some other not because maybe they don't exist in PLC. **Is important to check the single item Result**.

Remarks

To use WriteMultiVars a special class is supplied **S7MultiVar** that avoids the use of unsafe code (since there is a IntPtr)

Due the different kind of variables involved , there is no split feature available for this function, so **the maximum data size must not exceed the PDU size**

The advantage of this function becomes big when you have many small noncontiguous variables to be written.

Lean Data I/O functions

These are utility functions that simplify the use of ReadArea and WriteArea.
I.e. they call internally ReadArea and WriteArea passing the correct **Area** and **WordLen**.

Function	Purpose
DBRead	Reads a part of a DB from a PLC.
DBWrite	Writes a part of a DB into a PLC.
ABRead	Reads a part of IPU area from a PLC.
ABWrite	Writes a part of IPU area into a PLC.
EBRead	Reads a part of IPI area from a PLC.
EBWrite	Writes a part of IPI area into a PLC.

MBRead	Reads a part of Merkers area from a PLC.
MBWrite	Writes a part of Merkers area into a PLC.
TMRead	Reads timers from a PLC.
TMWrite	Write timers into a PLC.
CTRead	Reads counters from a PLC.
CTWrite	Write counters into a PLC.

Block oriented functions

Function	Purpose
GetAgBlockInfo	Returns info about a given block in PLC memory.
DBGet	Uploads a DB from the PLC
DBFill	Fills a DB into the PLC with a given value.

GetAgBlockInfo

Description

Returns some information about a given block.
This function is very useful if you need to read or write data in a DB which you do not know the size in advance (see **MC7Size** field).

This function is used internally by DBGet.

Declaration

```
public int GetAgBlockInfo(int BlockType, int BlockNumber,
  ref S7BlockInfo Block)
```

Parameters

	Type	Dir.	
BlockType	int	In	Type of Block that we need
BlockNum	int	In	Number of Block
Block	S7BlockInfo	in	S7BlockInfo struct

BlockType values

Helper Const	Type	Value
S7Consts.Block_OB	OB	0x38
S7 Consts.Block_DB	DB	0x41
S7 Consts.Block_SDB	SDB	0x42
S7 Consts.Block_FC	FC	0x43
S7 Consts.Block_SFC	SFC	0x44
S7 Consts.Block_FB	FB	0x45
S7 Consts.Block_SFB	SFB	0x46

S7BlockInfo struct

```
public struct S7BlockInfo {
    public int BlkType;    // Block Type (see SubBlkType table)
    public int BlkNumber;  // Block number
    public int BlkLang;    // Block Language (see LangType Table)
    public int BlkFlags;   // Block flags (bitmapped)
    public int MC7Size;    // The real size in bytes
    public int LoadSize;   // Load memory size
    public int LocalData;  // Local data
    public int SBBLength;  // SBB Length
    public int CheckSum;   // Checksum
    public int Version;    // Version (BCD 00<HI><LO>)
    public string;         // Code date
    public string;         // Interface date
    public string;         // Author
    public string;         // Family
    public string;         // Header
};
```

This struct is filled by the function, some fields require additional info:

SubBlockType table

	Value	Type
SubBlk_OB	0x08	OB
SubBlk_DB	0x0A	DB
SubBlk_SDB	0x0B	SDB
SubBlk_FC	0x0C	FC
SubBlk_SFC	0x0D	SFC
SubBlk_FB	0x0E	FB
SubBlk_SFB	0x0F	SFB

LangType table

	Value	Block Language
BlockLangAWL	0x01	AWL
BlockLangKOP	0x02	KOP
BlockLangFUP	0x03	FUP
BlockLangSCL	0x04	SCL
BlockLangDB	0x05	DB
BlockLangGRAPH	0x06	GRAPH

Return value

- 0 : The function was accomplished with no errors.
- Other values : see the Errors Code List.

DBGet

Description

Read an entire DB from the PLC without the need of specifying its size. As output *SizeRead* will contain the size read.

This function **is not subject to the security level set**.

Declaration

```
public int DBGet(int DBNumber, byte[] usrData, ref int Size)
```

Parameters

	Type	Dir.	
DBNumber	int	In	DB Number

Data	byte array	in	Address of the user buffer
SizeRead	int ref	In/Out	In : Buffer size supplied Out : Number of bytes read

Return value

- 0 : The function was accomplished with no errors.
- Other values : see the Errors Code List.

Remarks

This function first gathers the DB size via GetAgBlockInfo then calls ReadArea if the Buffer size is greater than the DB size, otherwise returns an error.

DBFill

Description

Fills a DB in AG with a given byte without the need of specifying its size.

Declaration

```
public int Cli_DBFill(int DBNumber, int FillChar);
```

Parameters

	Type	Dir.	
DBNumber	integer 32	In	DB Number
FillChar	Integer 32	in	Byte pattern

Return value

- 0 : The function was accomplished with no errors.
- Other values : see the Errors Code List.

Remarks

Fillchar is an integer for efficiency reasons, only the lowest byte is used.

Date/Time functions

These functions allow to read/modify the date and time of a PLC.

Imagine a production line in which each PLC saves the data with date/time field inside, it is very important that the date be up to date.

Both CP X43 and internal PN allow to synchronize date and time but you need an NTP server, and in some cases (old hardware or CP343-1 Lean or old firmware release) this doesn't work properly.

Snap7 Client, using the same method of S7 Manager, always works.

Function	**Purpose**

GetPlcDateTime	Returns the PLC date/time.
SetPlcDateTime	Sets the PLC date/time with a given value.
SetPlcSystemDateTime	Sets the PLC date/time with the host (PC) date/time.

GetPlcDateTime

Description

Reads PLC date and time into a C# DateTime class instance.

Declaration

```
public int GetPlcDateTime(ref DateTime DT)
```

Parameters

	Type	Dir.	
DateTime	Date	In	DateTime class instance

Return value

- 0 : The function was accomplished with no errors.
- Other values : see the Errors Code List.

SetPlcDateTime

Description

Sets the PLC date and time.

Declaration

```
public int SetPlcDateTime(DateTime DT)
```

Parameters

	Type	Dir.	
DateTime	Date	In	DateTime class instance

Return value

- 0 : The function was accomplished with no errors.
- Other values : see the Errors Code List.

SetPlcSystemDateTime

Description

Sets the PLC date and time in accord to the PC system Date/Time.

Declaration

```
public int SetPlcSystemDateTime()
```

Return value

- 0 : The function was accomplished with no errors.
- Other values : see the Errors Code List.

System info functions

these functions access to **SZL** (or **SSL** - System Status List) to give you all the same information that you can get from S7 Manager.

System Status List

The system status list (SSL) describes the current status of a programmable logic controller.

The contents of the SSL can only be read using information functions but cannot be modified. The partial lists are virtual lists, in other words, they are only created by the operating system of the CPUs when specifically requested.

You can access to system status list using **SFC 51** too "RDSYSST."

To read a partial list you must specify its **ID** and **Index**.

For a detailed description of SZL see:

§33 of **"System Software for S7-300/400 System and Standard Functions".**

Function	Purpose
ReadSZL	Reads a partial list of given ID and Index.
GetOrderCode	Returns the CPU order code.
GetCpuInfo	Returns some information about the AG.
GetCpInfo	Returns some information about the CP (communication processor).

ReadSZL

Description

Reads a partial list of given **ID** and **INDEX**.

Declaration

```
public int ReadSZL(int ID, int Index, ref S7SZL SZL, ref int Size)
```

Parameters

	Type	Dir.	
ID	int	In	List ID
Index	int	In	List Index
SZL	S7Szl	in	S7Szl class instance
Size	int	out	Size Read in bytes

S7SZL

```
[StructLayout(LayoutKind.Sequential, Pack = 1)]
 public struct SZL_HEADER
 {
     public UInt16 LENTHDR;
     public UInt16 N_DR;
 };

 [StructLayout(LayoutKind.Sequential, Pack = 1)]
 public struct S7SZL
 {
     public SZL_HEADER Header;
     [MarshalAs(UnmanagedType.ByValArray)]
     public byte[] Data;
 };
```

	Type	Dir.	Mean
LENTHDR	integer	Out	Length of a data record of the partial list in bytes
N_DR	integer	Out	Number of data records contained in the partial list.

Return value

- 0 : The function was accomplished with no errors.
- Other values : see the Errors Code List.

Remarks

Remember to allocate the Data[] buffer (see the demo)

GetOrderCode

Description

Gets CPU order code and version info.

Declaration

```
public int GetOrderCode(S7OrderCode Info)
```

Parameters

	Type	Dir.	
Info	S7OrderCode	Out	See below

```
public struct S7OrderCode
{
    public string Code;
    public byte V1; // Version 1st digit
    public byte V2; // Version 2nd digit
    public byte V3; // Version 3th digit
};
```

The Order code is a string such as “6ES7 151-8AB01-0AB0”

GetCpuInfo

Description

Gets CPU module name, serial number and other info.

Declaration

```
public int GetCpuInfo(S7CpuInfo Info)
```

Parameters

	Type	Dir.	
Info	S7CpuInfo	Out	See below

```
public struct S7CpuInfo
{
    public string ModuleTypeName;
    public string SerialNumber;
    public string ASName;
    public string Copyright;
    public string ModuleName;
}
```

Return value

- 0 : The function was accomplished with no errors.
- Other values : see the Errors Code List.

GetCpInfo

Description

Gets CP (communication processor) info.

Declaration

```
public int GetCpInfo(S7CpInfo Info)
```

Parameters

	Type	Dir.	
Info	S7CpInfo	Out	See below

S7CpInfo

```
public struct S7CpInfo
{
    public int MaxPduLength;
    public int MaxConnections;
    public int MaxMpiRate;
    public int MaxBusRate;
};
```

Return value

- 0 : The function was accomplished with no errors.
- Other values : see the Errors Code List.

PLC control functions

With these control function it's possible to Start/Stop a CPU and read the PLC status.

Function	Purpose
PlcColdStart	Puts the CPU in RUN mode performing an COLD START.
PlcHotStart	Puts the CPU in RUN mode performing an HOT START.
PlcStop	Puts the CPU in STOP mode.
PlcGetStatus	Returns the CPU status (running/stopped).

PlcColdStart

Description

Puts the CPU in RUN mode performing an COLD START.

Declaration

```
public int PlcColdStart()
```

Return value

- 0 : The function was accomplished with no errors.
- Other values : Either the PLC is already running or the current protection level is not met to perform this operation.

Remarks

This function is subject to the security level set.

PlcHotStart

Description

Puts the CPU in RUN mode performing an HOT START.

Declaration

```
public int PlcHotStart()
```

Return value

- 0 : The function was accomplished with no errors.
- Other values : Either the PLC is already running or the current protection level is not met to perform this operation.

Remarks

This function is subject to the security level set.

PlcStop

Description

Puts the CPU in STOP mode.

Declaration

```
public int PlcStop()
```

Return value

- 0 : The function was accomplished with no errors.
- Other values : Either the PLC is already stopped or the current protection level is not met to perform this operation.

Remarks

This function is subject to the security level set.

PlcGetStatus

Description

Returns the CPU status (running/stopped) into *Status.Value*.

Declaration

```
public int GetPlcStatus(ref int Status)
```

Parameters

	Type	Dir.	
Status	int	Out	Plc status

Status values

Helper Const	Value	
S7.S7CpuStatusUnknown	0x00	The CPU status is unknown.
S7.S7CpuStatusRun	0x08	The CPU is running.
S7.S7CpuStatusStop	0x04	The CPU is stopped.

Return value

- 0 : The function was accomplished with no errors.
- Other values : see the Errors Code List.

Security functions

With these functions is possible to know the current protection level, and to set/clear the current session password.

The correct name of the below functions SetSessionPassword and ClearSessionPassword, would have to be **Login** and **Logout** to avoid misunderstandings about their scope.

Especially because, if you look at the source code, there is an encoding function that translates the plain password before send it to the PLC.

PASSWORD HACKING IS VERY FAR FROM THE AIM OF THIS PROJECT, MOREOVER YOU NEED TO KNOW THE CORRECT PASSWORD TO MEET THE CPU SECURITY LEVEL.

Detailed information about the protection level can be found in §33.19 of **"System Software for S7-300/400 System and Standard Functions".**

Function	Purpose
SetSessionPassword	Send the password to the PLC to meet its security level.
ClearSessionPassword	Clears the password set for the current session (logout).
GetProtection	Gets the CPU protection level info.

SetSessionPassword

Description

Send the password to the PLC to meet its security level.

Declaration

```
public int SetSessionPassword(string Password);
```

Parameters

	Type	Dir.	
Password	String	In	8 chars UTF-8 string

Return value

- 0 : The function was accomplished with no errors.
- Other values : see the Errors Code List.

Remarks

A password accepted by a PLC is an 8 chars string, a greater password will be truncated, and a smaller one will be "right space padded".

ClearSessionPassword

Description

Clears the password set for the current session (logout).

Declaration

```
public int ClearSessionPassword()
```

Return value

- 0 : The function was accomplished with no errors.
- Other values : see the Errors Code List.

GetProtection

Description

Gets the CPU protection level info.

Declaration

```
public int GetProtection(ref S7Protection Protection)
```

Parameters

	Type	Dir.	
Protection	S7Protection	Out	See below

TS7Protection fields

```
public struct S7Protection
{
    public ushort sch_schal;
    public ushort sch_par;
    public ushort sch_rel;
    public ushort bart_sch;
    public ushort anl_sch;
};
```

Field Values

	Values	
sch_schal	1, 2, 3	Protection level set with the mode selector.
sch_par	0, 1, 2, 3	Password level, 0 : no password
sch_rel	0, 1, 2, 3	Valid protection level of the CPU
bart_sch	1, 2, 3, 4	Mode selector setting (1:RUN, 2:RUN-P, 3:STOP, 4:MRES, 0:undefined or cannot be determined)
anl_sch;	0, 1, 2	Startup switch setting (1:CRST, 2:WRST, 0:undefined, does not exist of cannot be determined)

See also §33.19 of "System Software for S7-300/400 System and Standard Functions"

Return value

- 0 : The function was accomplished with no errors.
- Other values : see the Errors Code List.

Properties/Info Functions

ConnTimeout

int, **in/Out** – Gets/Sets the connection timeout (in milliseconds) for a telegram.

RecvTimeout

int, **in/out** – Gets/Sets the receiving timeout (in milliseconds) for a telegram.

SendTimeout

int, **in/out** – Gets/Sets the sending timeout (in milliseconds) for a telegram.

Connected

bool, **out** – It’s true if the Client is connected.

PduSizeNegotiated

int, **out** – Returns the PDU size negotiated.

PduSizeRequested

int, **in/out** – Gets/Sets the PDU size to be negotiated.

PLCPort

int, **in/out** – Gets/Sets the connection Port (by default is 102).

ExecutionTime

int, **out** – Returns the last operation execution time in ms.

`public int LastError()`

Returns the last operation error.

`public string ErrorText(int Error)`

Returns a textual explanation of Error

`public int ExecTime()`

Returns the last operation execution time in ms. (compatibility with Snap7.net.cs)

S7 Helper reference

This class allow to read/write an “S7” value into a byte buffer, all new S71200/1500 types are supported.

Read functions are in format:

public static <.NET native Type> **Get**<S7 Type>**At**(`byte[]` Buffer, `int` Pos)

They extract a <S7 Type> var from the byte buffer starting from the byte number “pos”.

Write functions are in format:

public static void Set<S7 Type>**At**(`byte[]` Buffer, `int` Pos, <.NET native Type> Value)

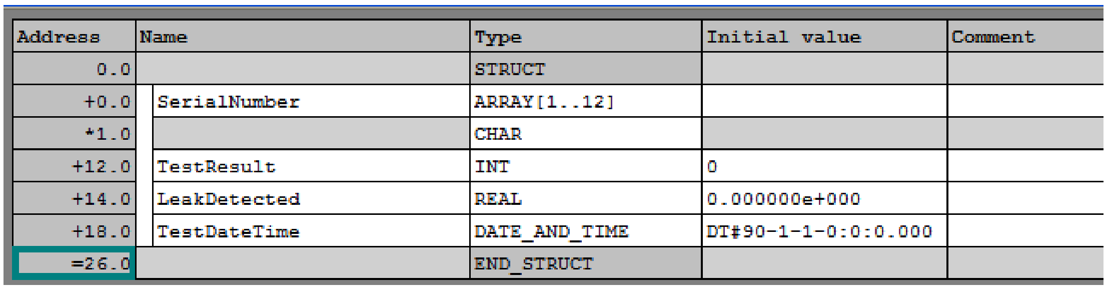
They insert a .net native type var into the byte buffer starting from the byte number “pos”.

Data access example

Now, just a real example of how to read a S7 struct from a PLC into .NET struct using this class in **C#** and **VB**.

Let’s suppose that we make a leak test on an automotive component and that we want to acquire the result of this test.

The data of this test consists of some values (Component serial number, Leak Value etc..) stored in DB 100 as in figure.



Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	SerialNumber	ARRAY[1..12]		
*1.0		CHAR		
+12.0	TestResult	INT	0	
+14.0	LeakDetected	REAL	0.000000e+000	
+18.0	TestDateTime	DATE_AND_TIME	DT#90-1-1-0:0:0.000	
=26.0		END_STRUCT		

Every time a component is tested this struct is filled by the PLC and we want to read it into a .NET struct, picking the fields and adjusting their byte-order (the PLC is Big-Endian, the PC is Little-Endian, so the bytes must be reversed).

This is our **C#** sample struct:

```
public struct ComponentResult
{
     public String SerialNumber;   // Component Serial Number
     public int TestResult;        // Result code 0:Unknown, 1:Good, 2:Scrap
     public double LeakDetected;   // Leak value [cc/min]
     public DateTime TestDateTime; // Test Timestamp
}
```

And this is the **C#** sample function that fills the struct. Notice that the outcome of any Client function call should be checked, here is skipped for brevity.

```
private ComponentResult LeakResult()
{
    ComponentResult Result = new ComponentResult();
    byte[] Buffer = new byte[26];
    // Reads the buffer.
    Client.DBRead(100, 0, 26, Buffer);
    // Extracts the fields and inserts them into the struct
    Result.SerialNumber = S7.GetCharsAt(Buffer, 0, 12);
    Result.TestResult = S7.GetIntAt(Buffer, 12);
    Result.LeakDetected = S7.GetRealAt(Buffer, 14);
    Result.TestDateTime = S7.GetDateTimeAt(Buffer, 18);
    return Result;
}
```

Same as above in **VB.NET**

The struct

```
Private Structure ComponentResult
    Public SerialNumber As String   ' Component Serial Number
    Public TestResult As Integer    ' Result code 0:Unknown, 1:Good, 2:Scrap
    Public LeakDetected As Double   ' Leak value [cc/min]
    Public TestDateTime As DateTime ' Test Timestamp
```

```
End Structure
```

…and the function

```
Private Function LeakResult() As ComponentResult
    Dim Result As ComponentResult
    Dim Buffer(26) As Byte
    Client.DBRead(100, 0, 26, Buffer)
    Result.SerialNumber = S7.GetCharsAt(Buffer, 0, 12)
    Result.TestResult = S7.GetIntAt(Buffer, 12)
    Result.LeakDetected = S7.GetRealAt(Buffer, 14)
    Result.TestDateTime = S7.GetDateTimeAt(Buffer, 18)
    Return Result
End Function
```

Finally, if you put a button and 4 labels in the form of VB demo, with this simple code you can test the mechanism.

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click

   Dim CR As ComponentResult = LeakResult()

   Label1.Text = CR.SerialNumber ' It's already a string
   Label2.Text = System.Convert.ToString(CR.TestResult)
   Label3.Text = System.Convert.ToString(CR.LeakDetected)
   Label4.Text = System.Convert.ToString(CR.TestDateTime)

 End Sub
```

S7MultiVar reference

This class allows to fill the S7DataItem[] and performs a ReadMultivar or WriteMultivar without using unsafe code.

The best way to understand the use of this class is analyzing an example.

The process is simple:

1. Instantiate the class.
2. Add the vars reference (i.e. your buffer) specifying the kind of element that we want read or write.
3. Call Read or Write method.

Read example

```
// Multi Reader Instance specifying Client
S7MultiVar Reader = new S7MultiVar(Client);

// Our buffers
byte[] DB_A = new byte[1024];
byte[] DB_B = new byte[1024];
byte[] DB_C = new byte[1024];

// Our DB number references
int DBNumber_A = 1; // DB1
int DBNumber_B = 1; // DB2
int DBNumber_C = 1; // DB3

// Add Items def. specifying 16 bytes to read starting from 0
```

```
Reader.Add(S7Consts.S7AreaDB, S7Consts.S7WLByte, DBNumber_A, 0, 16, ref DB_A);
Reader.Add(S7Consts.S7AreaDB, S7Consts.S7WLByte, DBNumber_B, 0, 16, ref DB_B);
Reader.Add(S7Consts.S7AreaDB, S7Consts.S7WLByte, DBNumber_C, 0, 16, ref DB_C);

// Performs the Read
int Result = Reader.Read();
```

If everything is ok into DB_A, DB_B, DB_C we will find the first 16 bytes of DB1, DB2 and DB3.

Remarks

1. Reader.Result[] will contain the single outcome of the items, it's possible that some items were read and some other not.
2. The parameters of Add() are the same of Read/WriteArea, you can read/write non only DB but also E/A/MK/T/C and also bits.

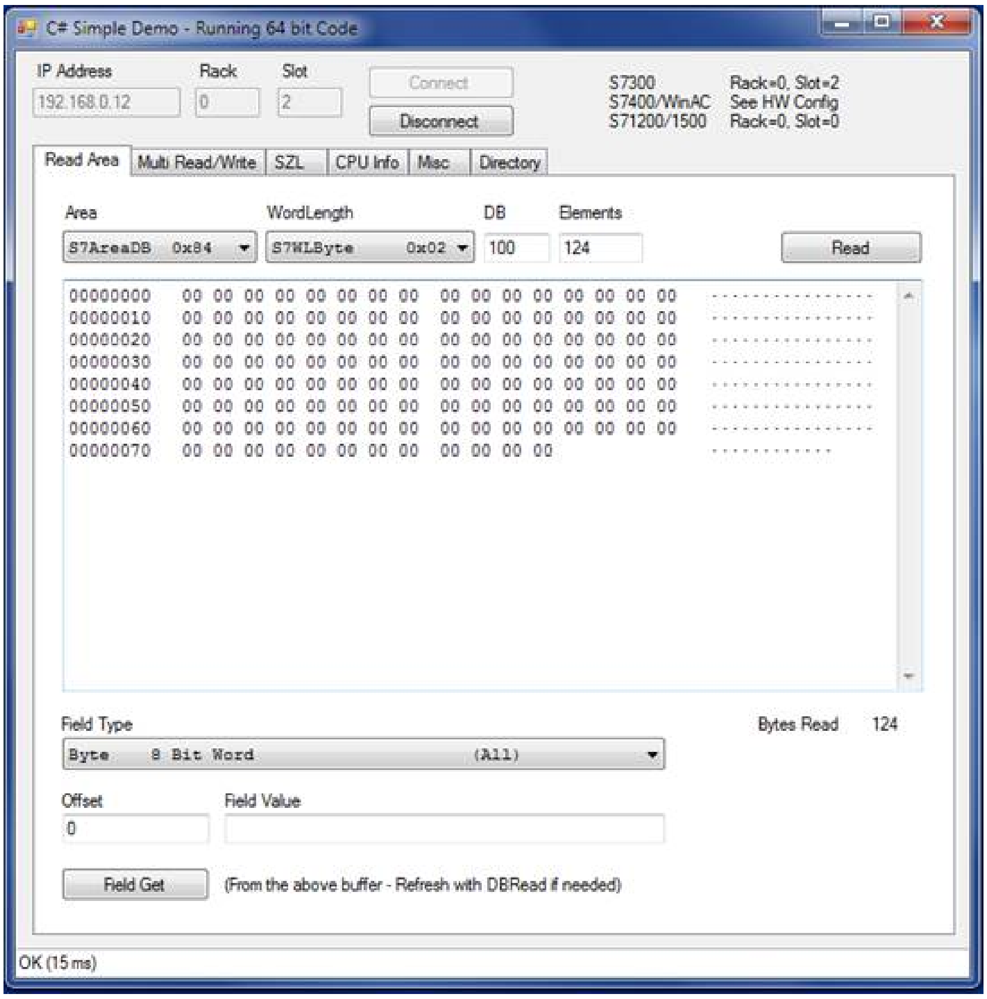
Error codes

Please refer to code of the function ErrorText() for the explanation

```
errTCPSocketCreation         = 0x00000001;
errTCPConnectionTimeout      = 0x00000002;
errTCPConnectionFailed       = 0x00000003;
errTCPReceiveTimeout         = 0x00000004;
errTCPDataReceive            = 0x00000005;
errTCPSendTimeout            = 0x00000006;
errTCPDataSend               = 0x00000007;
errTCPConnectionReset        = 0x00000008;
errTCPNotConnected           = 0x00000009;
errTCPUnreachableHost        = 0x00002751;
errIsoConnect                = 0x00010000;
errIsoInvalidPDU             = 0x00030000;
errIsoInvalidDataSize        = 0x00040000;
errCliNegotiatingPDU         = 0x00100000;
errCliInvalidParams          = 0x00200000;
errCliJobPending             = 0x00300000;
errCliTooManyItems           = 0x00400000;
errCliInvalidWordLen         = 0x00500000;
errCliPartialDataWritten     = 0x00600000;
errCliSizeOverPDU            = 0x00700000;
errCliInvalidPlcAnswer       = 0x00800000;
errCliAddressOutOfRange      = 0x00900000;
errCliInvalidTransportSize   = 0x00A00000;
errCliWriteDataSizeMismatch  = 0x00B00000;
errCliItemNotAvailable       = 0x00C00000;
errCliInvalidValue           = 0x00D00000;
errCliCannotStartPLC         = 0x00E00000;
errCliAlreadyRun             = 0x00F00000;
errCliCannotStopPLC          = 0x01000000;
errCliCannotCopyRamToRom     = 0x01100000;
errCliCannotCompress         = 0x01200000;
errCliAlreadyStop            = 0x01300000;
errCliFunNotAvailable        = 0x01400000;
errCliUploadSequenceFailed   = 0x01500000;
errCliInvalidDataSizeRecvd   = 0x01600000;
errCliInvalidBlockType       = 0x01700000;
errCliInvalidBlockNumber     = 0x01800000;
errCliInvalidBlockSize       = 0x01900000;
errCliNeedPassword           = 0x01D00000;
errCliInvalidPassword        = 0x01E00000;
```

```
errCliNoPasswordToSetOrClear = 0x01F00000;
errCliJobTimeout             = 0x02000000;
errCliPartialDataRead        = 0x02100000;
errCliBufferTooSmall         = 0x02200000;
errCliFunctionRefused        = 0x02300000;
errCliDestroying             = 0x02400000;
errCliInvalidParamNumber     = 0x02500000;
errCliCannotChangeParam      = 0x02600000;
errCliFunctionNotImplemented = 0x02700000;
```

Images



IP Address: 192.168.0.12 Rack: 0 Slot: 2

S7300 Rack=0, Slot=2
S7400/WinAC See HW Config
S71200/1500 Rack=0, Slot=0

Area: S7AreaDB 0x84 WordLength: S7WLByte 0x02 DB: 100 Elements: 124

```
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000070  00 00 00 00 00 00 00 00  00 00 00 00              ............
```

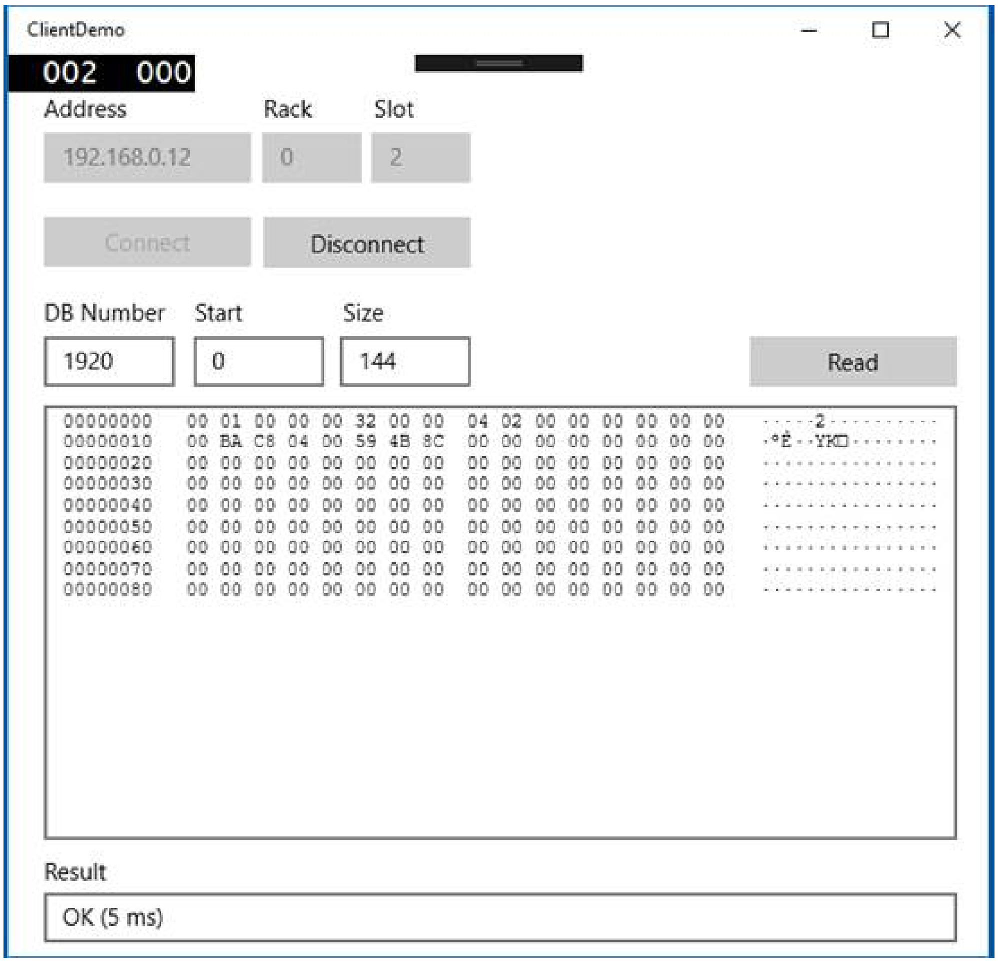
Field Type: Byte 8 Bit Word (All) Bytes Read 124

Offset: 0 Field Value:

(From the above buffer - Refresh with DBRead if needed)

OK (15 ms)

Demo WinForm C# Windows 7



Address: 192.168.0.12
Rack: 0
Slot: 2

DB Number: 1920
Start: 0
Size: 144

```
00000000   00 01 00 00 00 32 00 00  04 02 00 00 00 00 00 00   ·····2··········
00000010   00 BA C8 04 00 59 4B 8C  00 00 00 00 00 00 00 00   ·°È··YK□········
00000020   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ················
00000030   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ················
00000040   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ················
00000050   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ················
00000060   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ················
00000070   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ················
00000080   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ················
```

Result: OK (5 ms)

Demo Universal Windows Platform C# Windows 10