Settimino

Reference manual

Davide Nardella

Rev.2 – December 09, 2016

Summary

Project overview

Settimino is an open source Ethernet library for interfacing **ARDUINO™** or **NodeMCU™** natively with Siemens **S7 PLCs**. The CPUs 1200/1500 LOGO 0BA7 and S7200 are also supported.

Settimino is not strictly "the porting" of Snap7 for ARDUINO because, although it runs fine in small boards such Raspberry™, its design (multiplatform 32/64 bit) is too exacting for a tiny 8 bit board. I rewrote the Snap7 micro-client by optimizing its footprint to be suitable with Arduino.

Main features

- Full PLC memory access.
- PDU independence, i.e. the data that you can transfer in a single call depends only on the Arduino memory availability.
- Helper functions for data conversion (Big Endian --> Little endian)
- Three memory models for footprint optimization.
- Uses standard Arduino Ethernet library i.e. it can coexist with other clients/servers in the same sketch.
- 3 ms to read a PDU into the internal buffer, 24 ms for 1024 bytes into the external.

What do you need

To work with Settimino you need (see § Compatibility for details):

- An **ARDUINO** equipped with an Ethernet shield (with W5100) in which to load the examples supplied.

- A Siemens PLC equipped with Ethernet port or Communication Processor.
  Any PLC is supported except for the old S5 family.
  Or, if you don't have it, download **Snap7** project (see below).

The Snap7 Library is not strictly required but I suggest you to download it (it's free of course) because it contains:

- A ready to run Client Demo (Win/Unix/BSD/Solaris) that allows you to see if what you are transferring is correct without the needing of Simatic Manager.
- A ready to run Server Demo (same platforms) that emulates a S7300 PLC equipped with Ethernet adapter **that allows you to test your Settimino based applications without having a physical PLC**.

Even this documentation is derived from that of Snap7, visit its homepage for a more detailed information.

See also § **Testing Platforms**.

Licensing

Settimino 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 Settimino 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 QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IF ANYONE BELIEVES THAT, WITH SETTIMINO PROJECT HAVE BEEN VIOLATED SOME COPYRIGHTS, PLEASE EMAIL US, AND ALL THE NECESSARY CHANGES WILL BE MADE.

Settimino reference

Settimino is an ARDUINO library, it consists of three files

1. Settimino.h
2. Settimino.cpp
3. Keywords.txt

It depends on the supplied Ethernet Library (<Arduino Dir>\libraries\Ethernet).

To use it, just like other libraries, you need to import it in your programming environment.

To do this follow this excellent tutorial
http://arduino.cc/en/Guide/Libraries

Instead, to modify/extend Settimino follow
http://arduino.cc/en/Hacking/LibraryTutorial

S7Client

The main class exported is **S7Client** which is the object that is able to connect and transfer the data from Arduino to a Siemens PLC (and vice-versa).

There is not a global instance of S7Client, you have to instance it (or them) in your sketch.

Each client uses a socket (via the EthernetClient object) so you can instantiate up to four clients or less if you are instantiating other Ethernet object (or derived) in the same project.
This is the limit of the W5100 Ethernet chip.

S7Client is a class, but as you can see in other libraries, the Object Oriented Paradigm must be revisited for Arduino, due the needing to have a small footprint for the programs, so let's see the Settimino choices.

- The Arduino environment is not multitask so, although you have 4 clients, only one at time is the active one.
- The client is fully synchronous, each job (transaction) is completely executed in its function call.

Said that, a global data buffer is used. It's shared across all clients since only one at time can use it.

This data buffer is able to contain only one PDU see Siemens communication to understand what this means. Substantially, the PDU is the base "data slice" handled by the PLC, if you need to transfer more data, the S7 protocol states that subsequent transactions (each containing a PDU) must be used.

The PDU variable is itself global, i.e. is visible from your sketch.

So you have two choice for optimizing the memory usage.

1. If you need to transfer more than 222 byte (the payload of a PDU) you need an external buffer.
2. If you want to transfer small amount of data, you can use an external buffer or use directly the PDU variable.

Let's see an example (please refer to the S7Client functions syntax for further information).

This is the PDU as defined in Settimino.h:

```
typedef struct{
  byte H[Size_WR];                       // PDU Header
  byte DATA[MaxPduSize-Size_WR+Shift];   // PDU Data
}TPDU;

TPDU PDU; // Declaration
```

The **H** array contains the protocol header, the **DATA** array contains the payload i.e. the raw data of the telegram.

```
// Large Data transfer

byte MyBuffer[1024]; // 1K array

Client.ReadArea(S7AreaDB, 24, 0, 1024, &MyBuffer);
```

We are saying to the Client to access to **DB 24,** to read **1024** bytes starting from the first (0) and to put them into **MyBuffer**.

```
// Small Data transfer

Client.ReadArea(S7AreaDB, 24, 0, 64, NULL);
Serial.println(PDU.DATA[0]); // Print the first byte received
Serial.println(PDU.DATA[1]); // Print the second… and so on
```

We are saying to the Client to access to **DB 24,** read **64** bytes starting from the first (0) and **to leave them** into the internal buffer.

Similarly for write operations, you put your data into a buffer or into PDU.DATA[] then call WriteArea().

Note:

The Client will safely **trim** the **Amount** parameter if you pass NULL as buffer pointer and Amount is greater than the size of PDU.DATA[].

S7Helper

Siemens PLC data are Big-Endian, Arduino data are Little-Endian.

S7Helper is a global object that allows to extract right-formatted values from a byte buffer. The instance name is **S7**.

Let's suppose to upload the following DB 100.

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	PRESSURE	REAL	0.000000e+000	TRANSDUCER TP1
+4.0	ENCODER	DWORD	DW#16#0	MOTOR ENCODER
+8.0	COMP_NUMBER	INT	0	COMPONENT NUMBER
=10.0		END_STRUCT		

```
float Pressure;
unsigned long Encoder;
int16_t Component;
// Get 10 bytes starting from 0
Client.ReadArea(S7AreaDB, 100, 0, 10, &MyBuffer);
```

MyBuffer now will contain a "disordered" series of bytes (Big-Endian).

To access the fields we can write:

```
Pressure = S7.FloatAt(&MyBuffer, 0);
Encoder = S7.DWordAt(&MyBuffer, 4);
Component = S7.IntegerAt(&MyBuffer, 8);
```

S7Helper has two overloaded methods for each field access, so

```
Encoder = S7.DWordAt(4);
```

will refer to PDU.DATA[4]

Memory models

For further optimizing the footprint, three memory models are available : **Small**, **Normal** and **Extended**.

Although the extended model runs fine in the ARDUINO UNO, you may want to further reduce the memory usage "by skipping" some features that you don't need using in your project.

For example, Settimino Client can control the PLC putting it in STOP or RUN mode, but if you only need to read/write data, maybe you don't need of these features.

To set the memory model you need simply to modify the directive at beginning of Settimino.h, writing:

```
#define _SMALL
```
or
```
#define _NORMAL
```
or
```
#define _EXTENDED
```

The "small" model contains the minimum set of functions to connect to a PLC and reading/writing data.

The "normal" mode contains the S7Helper.

Finally the "Extended" model contains also the remaining extended functions.

By default Settimino is released as **Extended**.

These directives are visible from your sketch.

Siemens S7 Protocol

If you already know the Siemens Ethernet communication you can skip this chapter.

Settimino, just like Snap7, by design, it only handles **Ethernet** S7 Protocol communications.

S7 Protocol, is the backbone of the Siemens communications, its Ethernet implementation relies on ISO TCP (RFC1006) which, by design, is block oriented.

Each block is named **PDU** (Protocol Data Unit), its maximum length depends on the CP and is negotiated during the connection.

S7 Protocol is **Function oriented** or **Command oriented**, i.e. each transmission contains a command or a reply to it.
If the size of a command doesn't fit in a PDU, then it's split across more subsequent PDU.

Each command consists of

- A header.
- A set of parameters.
- A parameters data.
- A data block.

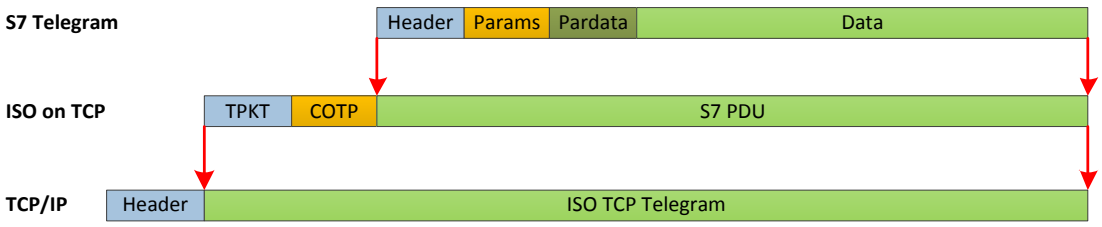
The first two elements are always present, the other are optional.

To understand:

***Write** this **data** into **DB 10** starting from the offset **4**.*
Is a command.

Write, DB, 10, 4 and data are the components of the command and are formatted in a message in accord to the protocol specifications.

S7 Protocol, ISO TCP and TCP/IP follow the well-known encapsulation rule : every telegram is the "payload" part of the underlying protocol.



Protocols Encapsulation

S7 Commands are divided into categories:

- **Data Read/Write**
- **Cyclic Data Read/Write**
- **Directory info**
- **System Info**
- **Blocks move**
- **PLC Control**
- **Date and Time**
- **Security**
- **Programming**

Siemens provides a lot of FB/FC (PLC side), **Simatic NET** software (PC side) and a huge excellent documentation about their use, but no internal protocol specifications.

PDU independence

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.

All Settimino functions completely hide this concept, the data that you can transfer in a single call depends only on the size of the available memory.

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

Compatibility

Arduino

Settimino was tested with these Arduino™ original shields:

- ARDUINO UNO R3
- ARDUINO MEGA 2560 R3

both equipped with

- ARDUINO Ethernet Shield R3

It should be compatible with other Arduino boards using a fully compatible Ethernet library that exposes the **EthernetClient** object.

Test reporting are welcomed...

NodeMCU ESP 12-E V1.0

Settimino PLC functions compatibility list

	CPU						
	300	**400**	**WinAC**	**1200**	**1500**	**LOGO**	**S7200**
DB Read/Write	**O**	**O**	**O**	**O(1)**	**O(1)**	**O(2)**	**O(2)**
EB Read/Write	**O**	**O**	**O**	**O**	**O**		
AB Read/Write	**O**	**O**	**O**	**O**	**O**		
MK Read/Write	**O**	**O**	**O**	**O**	**O**		
TM Read/Write	**O**	**O**	**O**	**-**	**-**	**-**	**-**
CT Read/Write	**O**	**O**	**O**	**-**	**-**	**-**	**-**
Get DB Size	**O**	**O**	**O**	**-**	**-**	**-**	**-**
DB Get	**O**	**O**	**O**	**-**	**-**	**-**	**-**
Control Run/Stop	**O**	**O**	**O**	**-**	**-**	**-**	**-**

(1) See S71200/1500 Notes
(2) The entire memory is mapped in the V Area accessible as DB 1 from the outside.

S7 300/400/WinAC


No special consideration has to be made about these CPU, Settimino has full access to these PLC either directly (with the integrated interface 3xx-PN or 4xx-PN) or via the CPX43 interface.

Connection

Use **ConnectTo()** specifying **IP_Address, Rack, Slot** for the first connection, this functions set the internal parameters and connects to the PLC. if a TCP error occurred and a disconnection was needed, for reconnecting you can simply use **Connect()** which doesn't requires any parameters. Look at the reference of ConnectTo() for a detailed explanation of Rack and Slot.

It's possible but it's not mandatory to specify the connection type via the function **SetConnectionType()** which must be called before ConnectTo(). By default the client connects as a **PG** (the programming console), with this function is possible change the connection resource type to **OP** (the Siemens HMI panel) or **S7 Basic** (a generic data transfer connection).
In the hardware configuration (Simatic Manager) of the CPU, under "Communication" tab, you can change, PLC-side, the connection's distribution, if you need.
PG, OP and S7 Basic communications are client-server connections, i.e. they don't require that the PLC have a connection designed by NetPro.
Note : This is an optimization function, if the client doesn't connect, the problem is **elsewhere**.

Connection type table

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

It's possible, but uncomfortable, to connect to a S7300/400 PLC using TSAPs (SetConnectionParams() function) as well.
To do this, use **0x0100** as Local TSAP and follow the next formula for the Remote TSAP.

```
RemoteTSAP=(ConnectionType<<8)+(Rack*0x20)+Slot;
```

S7 1200/1500


An external equipment can access to S71200/1500 CPU using the S7 "base" protocol, only working as an HMI, i.e. only basic data transfer are allowed.

All other PG operations (control/directory/etc..) must follow the extended protocol.

Connection

To connect with these PLC use **ConnectTo()** just like the other "S7" CPUs.
The only difference is that Rack and Slot are fixed (Rack=0, Slot=0).

Also **SetConnectionType()** can be used.

Data Access

To access a DB in S71500 some additional setting plc-side are needed.

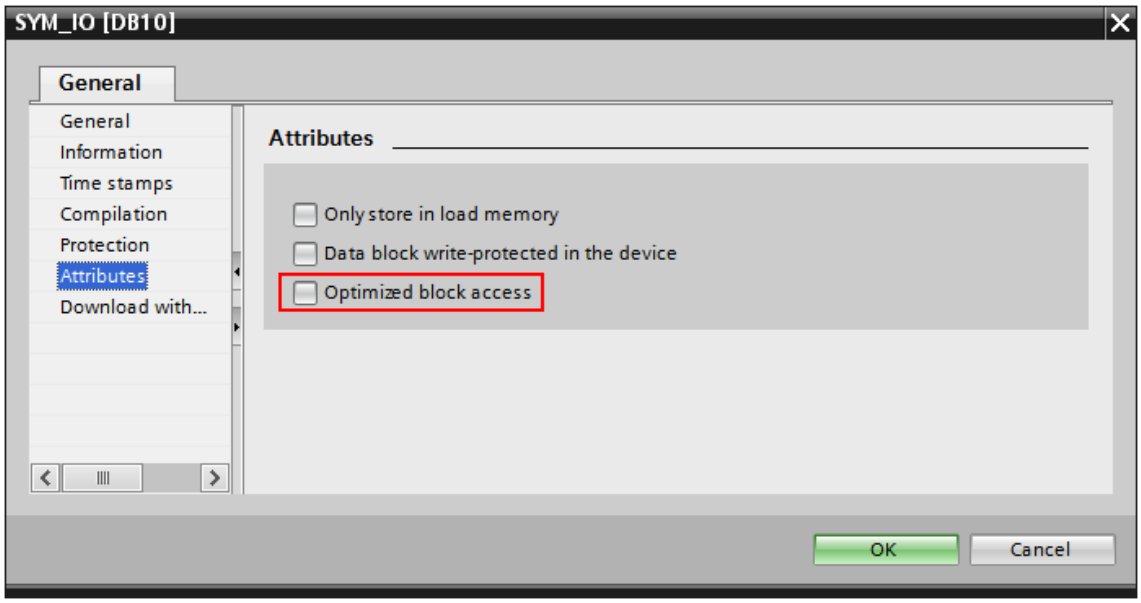
1. Only global DBs can be accessed.
2. The optimized block access must be turned off.
3. The access level must be "full" and the "connection mechanism" must allow GET/PUT.

Let's see these settings in TIA Portal V12

DB property

Select the DB in the left pane under "Program blocks" and press Alt-Enter (or in the contextual menu select "Properties…")

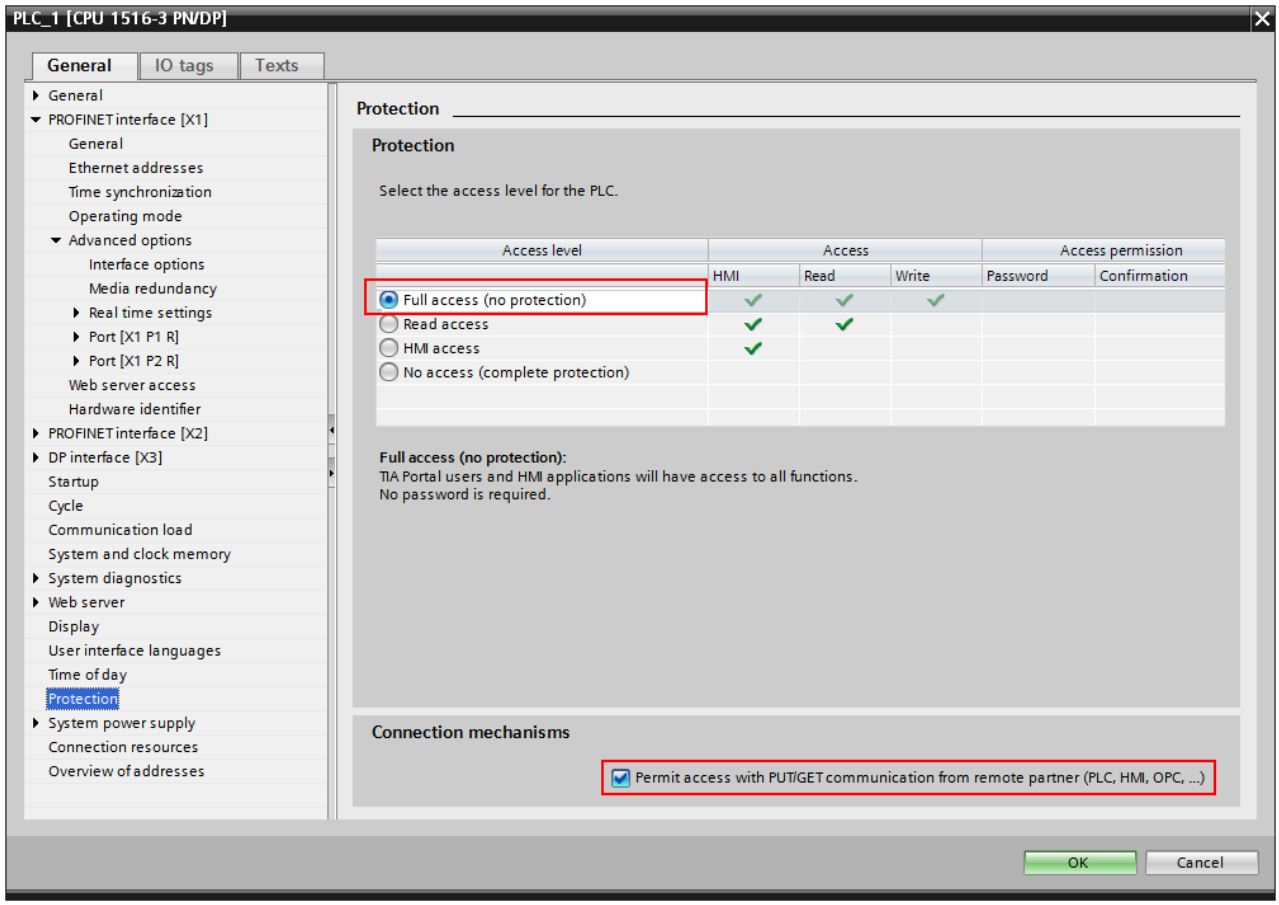
Uncheck Optimized block access, by default it's checked.



Protection

Select the CPU project in the left pane and press Alt-Enter (or in the contextual menu select "Properties…")

In the item Protection, select "Full access" and Check "Permit access with PUT/GET …." as in figure.



LOGO! 0BA7

LOGO is a small Siemens PLC suited for implementing simple automation tasks in industry and building management systems.

It's very user friendly and the last model is equipped with an Ethernet port for both programming and data exchange.

Communication

Due to its architecture, the LOGO communication is different from its Siemens cousins.

It implements two Ethernet protocols, the first that we can call **PG protocol,** is used by the software LOGO Comfort (the developing tool) to perform system tasks such as program upload/download, run/stop and configuration.
The second, used for data exchange, is the well-known (from the Settimino point of view) S7 Protocol.

They are very different, and the first is not covered by Snap7 and Settimino because is a stand-alone protocol that is not present, Is far I know, in different contexts.

To communicate with LOGO, the Ethernet connections must be designed with LOGO Comfort in advance.
Of course I will show you how.

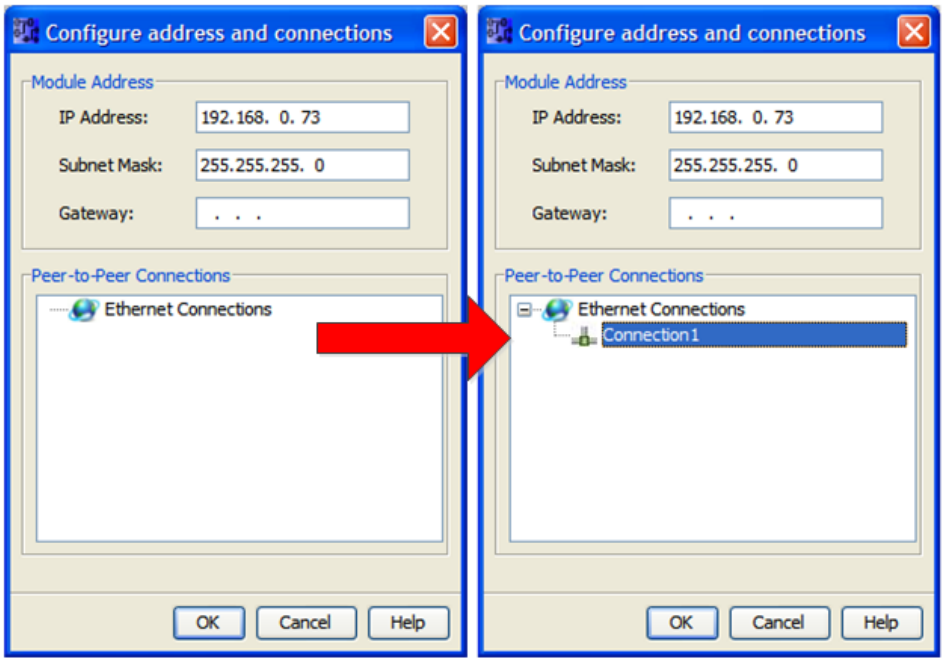
LOGO must be set as MASTER (i.e. NORMAL mode as LOGO Comfort says).

I assume that your LOGO Comfort is already set and connected to the LOGO.

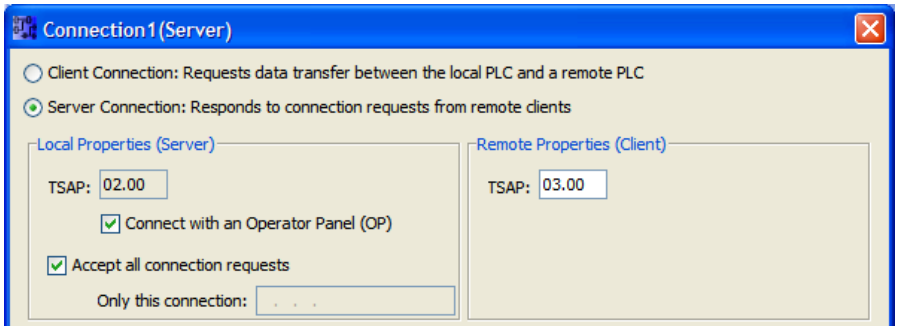
Connection configuration

Configuring a **server connection** allows you to connect Settimino Client with LOGO for reading and writing the memory just like an HMI panel would do.

- In the **Tools** menu choose the **Ethernet Connections** item.
- Right click on ”Ethernet Connections” and click ”Add connections” to add a connection



- Double-click the new connection created and edit its parameters selecting **Server Connection**.

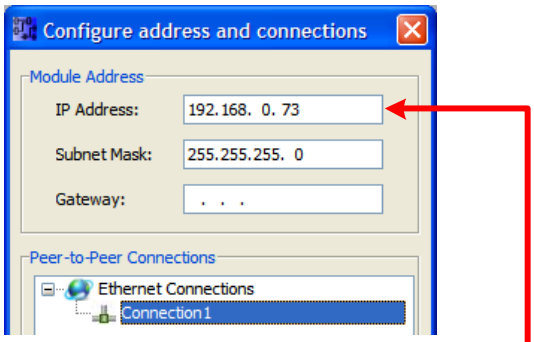


Note:

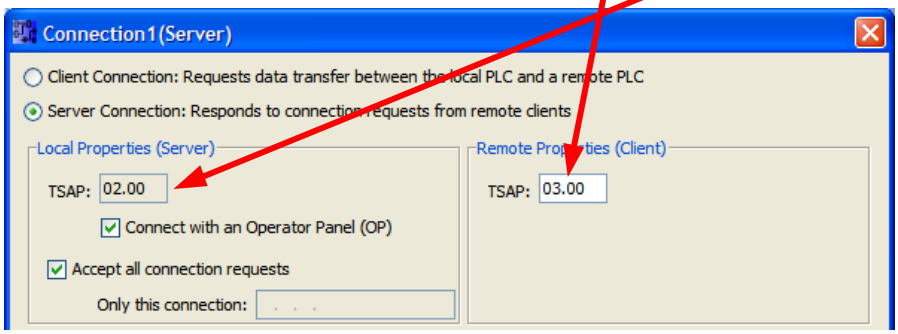
1. “Connect with an operator panel” checkbox can be checked or unchecked.
2. If you uncheck “Accept all connections” you must specify the PC address (for now I suggest you to leave it checked).

You can chose for Remote TSAP the same value of the Local TSAP, in the example I used two different values to remark (as you will see) the **crossing parameters**.

- Confirm the dialog, close the connection editor and **download** the configuration into the LOGO.
- The LOGO is ready, to test it run the ReadDemo, insert the LOGO IP Address and modify the connection routine as in figure.




```
IPAddress Peer(192,168,0,73);
Client.SetConnectionParams(Peer, 0x0300, 0x0200);
Client.Connect();
```



Notice that the Local TSAP of the Client corresponds to the Remote TSAP of the LOGO and vice-versa. This is the key concept for the S7 connections.

The LOGO memory that we can Read/Write is the **V** area that is seen by all HMI (and Snap7 too) as **DB 1.**
Into it are mapped all LOGO resources organized by bit, byte or word.

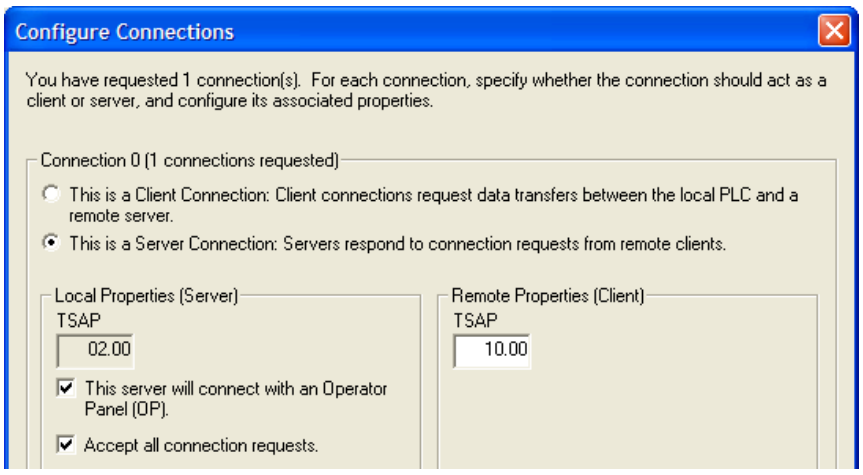
There are several tutorials in the Siemens site that show how to connect an HMI (via WinCC flexible or TIA) to the LOGO and the detailed map.
Please refer to them for further information.

S7 200 (via CP243)

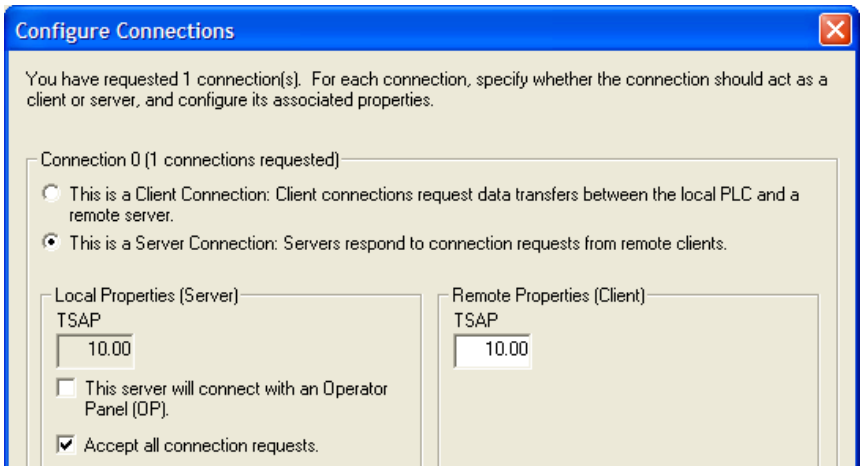
S7200 was out of the scope for Snap7 and Settimino because beginning November 2013 the S7-200 product family will enter into the Phase Out stage of its product life cycle, but after working with LOGO also this PLC can be accessed since it uses the same connection mechanism.

Consider experimental the support for this PLC

As said, the connection is very similar to that of LOGO, you need to design a connection using the Ethernet wizard of MicroWin as in figure.



or



In the first case the PLC expects to be connected to an OP and you must supply LocalTSAP = **0x1000** and RemoteTSAP = **0x0200** to the SetConnectionParams function.

If you make a S7200 HMI project, the runtime of WinCC itself uses these parameters.

In the second case you should use LocalTSAP = **0x1000** and RemoteTSAP = **0x1000.**

S7Client syntax

Administrative functions

(Memory Model : Small/Normal/Extended)

These functions 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

```
void SetConnectionType(uint16_t ConnectionType);
```

Parameters

	Type	Dir.	
ConnectionType	uint16_t	In	See the table

Connection type table

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

ConnectTo

Description

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

Declaration

```
int ConnectTo(IPAddress Address, uint16_t Rack, uint16_t Slot);
```

Parameters

	Type	Dir.	
Address	IPAddress class	In	PLC/Equipment IPV4 Address ex. 192,168,1,12
Rack	uint16_t	In	PLC Rack number (see below)
Slot	uint16_t	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.

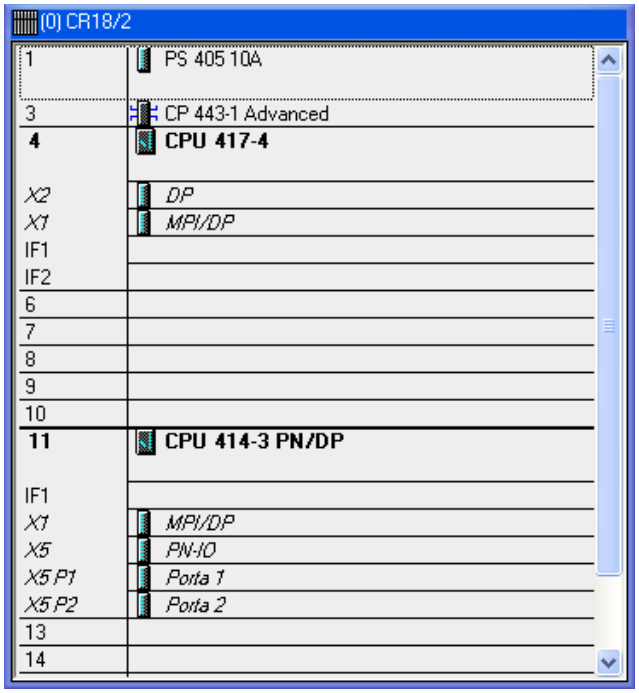
Rack and Slot

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.

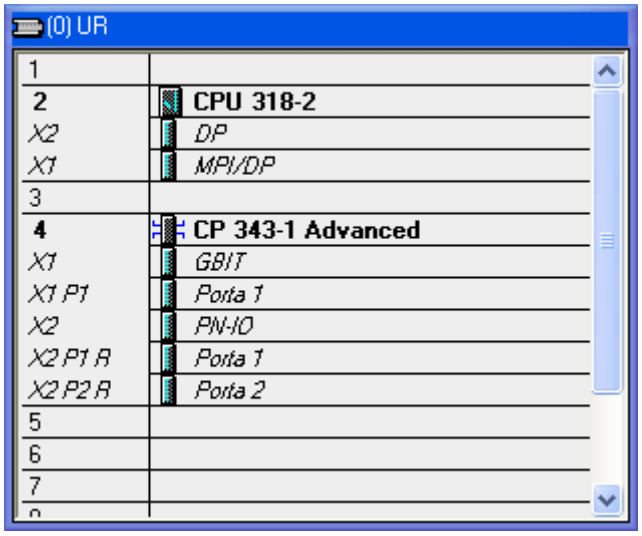
Let's see some examples of hardware configuration:



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.



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

```
void SetConnectionParams(IPAddress Address,uint16_t LocalTSAP,
   uint16_t RemoteTSAP);
```

Parameters

	Type	Dir.	
Address	IPAddress class	In	PLC/Equipment IPV4 Address ex. 192,168,1,12
LocalTSAP	uint16_t	In	Local TSAP (Arduino TSAP)
RemoteTSAP	uint16_t	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

```
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

```
void Disconnect();
```

Remarks

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

Data I/O functions

(Memory Model : Small/Normal/Extended)

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.

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

```
int ReadArea(int Area, uint16_t DBNumber, uint16_t Start,
    uint16_t Amount, void *pUsrData);
```

Parameters

	Type	Dir.	Mean
Area	int	In	Area identifier.
DBNumber	uint16_t	In	DB Number if Area = S7AreaDB, otherwise is ignored.
Start	uint16_t	In	Offset to start
Amount	uint16_t	In	Amount of words to read (1)
pUsrData	Pointer to memory area	In	Address of user buffer. (see remarks)

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

Area table

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

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 is 240 bytes for this implementation of Settimino.

If this data size exceeds the PDU size, the packet is automatically split across more subsequent transfers and your buffer should be large enough to receive the data.

If NULL is passed as pUsrData, the data remains into the PDU buffer and the size requested is trimmed to the max (222 bytes).

This allow to save memory avoiding to supply a further memory buffer if you plan to make only small data transfers.

The parameter Amount is not the size in bytes.

Particularly:

Buffer size (byte) = Word size * Amount

Where:

	Word size
(E/A/M/DB) - Byte	1
Counter	2
Timer	2

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 buffer pointed by pUsrData into PLC.

Declaration

```
int WriteArea(int Area, uint16_t DBNumber, uint16_t Start,
    int Amount, void *pUsrData);
```

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

Block oriented functions

(Memory Model Extended)

Function	Purpose
GetDBSize	Returns the size of a given DB Number in AG.
DBGet	Uploads a DB from AG.

GetDBSize

Description

Returns the size of a given DB Number.
This function is useful to upload an entire DB.

Declaration

```
int GetDBSize(uint16_t DBNumber, uint16_t *Size);
```

Parameters

	Type	Dir.	
DBNumber	uint16_t	In	DB Number
size	Pointer to uint16_t	Out	DB Size in bytes

Return value

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

DBGet

Description

Uploads a DB from AG.

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

Only data is uploaded.

Declaration

```
int DBGet(uint16_t DBNumber, void *pUsrData, uint16_t *Size);
```

Parameters

	Type	Dir.	
DBNumber	uint16_t	In	DB Number
pUsrData	Pointer	in	Address of the user buffer
size	Pointer to uint16_t	In	Buffer size available
		Out	Bytes uploaded

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 GetDBSize then calls ReadArea if the Buffer size is greater than the DB size, otherwise returns an error.

This is an utility function implemented as above.

```
int S7Client::DBGet(uint16_t DBNumber, void *ptrData, uint16_t *Size)
{
  uint16_t Length;
  int Result;

  Result=GetDBSize(DBNumber, &Length);
  if (Result==0)
  {
    if (Length<=*Size) // Check if the buffer supplied is big enough
    {
       Result=ReadArea(S7AreaDB, DBNumber, 0, Length, ptrData);
       if (Result==0)
         *Size=Length;
    }
    else
      Result=errBufferTooSmall;
  }
  return Result;
}
```

PLC control functions

(Memory Model Extended)

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

Function	Purpose
PlcStart	Puts the CPU in RUN mode performing an HOT START.
PlcStop	Puts the CPU in STOP mode.
GetPlcStatus	Returns the CPU status (running/stopped).

PlcStart

Description

Puts the CPU in RUN mode performing an HOT START.

Declaration

```
int PlcStart();
```

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

```
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.

GetPlcStatus

Description

Returns the CPU status (running/stopped).

Declaration

```
int GetPlcStatus(int *Status);
```

Parameters

	Type	Dir.	
Status	Pointer to Integer	In	Address of Status variable.

Status values

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

Return value

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

Low level functions

(Memory Model Extended)

Settimino hides the IsoTCP underlying layer. With this function however, it's possible to exchange an ISO/TCP telegram with a PLC.

Function	Purpose
IsoExchangeBuffer	Exchanges a given S7 PDU (protocol data unit) with the CPU.

IsoExchangeBuffer

Description

Exchanges a given S7 PDU (protocol data unit) with the CPU.

Declaration

```
int IsoExchangeBuffer(uint16_t *Size);
```

Parameters

	Type	Dir.	
Size	Pointer to unsigned 16 bit integer	In	Buffer size
		Out	Reply telegram size

Return value

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

Remarks

The S7 PDU must be present into PDU variable.

No check is performed.

Miscellaneous functions

(Memory Model Small/Normal/Extended)

These are utility functions.

Function	Purpose
GetPduLength	Returns info about the PDU length (requested and negotiated).

GetPduLength

Description

Returns the value of the PDU length negotiated.

Declaration

```
int GetPduLength();
```

Return value

- 0 : The client is not connected.
- Other values : the PDU length.

Remarks

During the S7 connection Client and Server (the PLC) negotiate the PDU length.

Properties

RecvTimeout

uint16_t, **input** – It's the receiving timeout (in milliseconds) for a telegram.

LastError

int, **output** – Contains the last operation error.

Connected

bool, **output** – It's true if the Client is connected.

S7Helper syntax

(Memory Model : Normal/Extended)

Function	Purpose
BitAt	Returns the Bit at given location in an user buffer or in the PDU buffer.
ByteAt	Returns the Byte at given location in an user buffer or in the PDU buffer.
WordAt	Returns the Word at given location in an user buffer or in the PDU buffer.
DWordAt	Returns the DWord at given location in an user buffer or in the PDU buffer.
FloatAt	Returns the Float at given location in an user buffer or in the PDU buffer.
IntegerAt	Returns the Int at given location in an user buffer or in the PDU buffer.
DIntAt	Returns the DInt at given location in an user buffer or in the PDU buffer.

BitAt

Description

Returns the Bit at given location in an user buffer or in the PDU.DATA buffer.

Declaration

```
bool BitAt(void *Buffer, int ByteIndex, byte BitIndex);
bool BitAt(int ByteIndex, byte BitIndex);
```

Parameters

	Type	Dir.	
Buffer	Pointer	In	Pointer to an user buffer
ByteIndex	int	In	Start byte
BitIndex	byte	In	Bit inside the byte

Example

```
IsOn=S7.BitAt(&MyBuffer, 140, 2);
```

"IsOn" is true if the third bit of the byte 140 of MyBuffer is 1.

```
IsOn=S7.BitAt(140, 2);
```

"IsOn" is true if (PDU.DATA[140] & 0x04)!=0

ByteAt

Description

Returns the Byte at given location in an user buffer or in the PDU.DATA buffer.

Declaration

```
byte ByteAt(void *Buffer, int Index);
byte ByteAt(int Index);
```

Parameters

	Type	Dir.	
Buffer	Pointer	In	Pointer to an user buffer
Index	int	In	Start byte

Example

```
MyByte=S7.ByteAt(&MyBuffer, 140);
```

WordAt

Description

Returns the Word (16 bit unsigned integer) at given location in an user buffer or in the PDU.DATA buffer.

Declaration

```
word WordAt(void *Buffer, int Index);
word WordAt(int Index);
```

Parameters

	Type	Dir.	
Buffer	Pointer	In	Pointer to an user buffer
Index	int	In	Start byte

Example

```
MyWord=S7.WordAt(&MyBuffer, 140);
```

DWordAt

Description

Returns the DWord (32 bit unsigned integer) at given location in an user buffer or in the PDU.DATA buffer.

Declaration

```
dword DWordAt(void *Buffer, int Index);
dword DWordAt(int Index);
```

Parameters

	Type	Dir.	
Buffer	Pointer	In	Pointer to an user buffer
Index	int	In	Start byte

Example

```
MyDWord=S7.DWordAt(&MyBuffer, 140);
```

FloatAt

Description

Returns the Float (32 bit floating point number) at given location in an user buffer or in the PDU.DATA buffer.

Declaration

```
float FloatAt(void *Buffer, int Index);
float FloatAt(int Index);
```

Parameters

	Type	Dir.	
Buffer	Pointer	In	Pointer to an user buffer
Index	int	In	Start byte

Example

```
MyFloat=S7.FloatAt(&MyBuffer, 140);
```

IntegerAt

Description

Returns the Integer (16 bit signed integer) at given location in an user buffer or in the PDU.DATA buffer.

Declaration

```
integer IntegerAt(void *Buffer, int Index);
integer IntegerAt(int Index);
```

Parameters

	Type	Dir.	
Buffer	Pointer	In	Pointer to an user buffer
Index	int	In	Start byte

Example

```
MyInteger=S7.IntegerAt(&MyBuffer, 140);
```

DIntAt

Description

Returns the Double Integer (32 bit signed integer) at given location in an user buffer or in the PDU.DATA buffer.

Declaration

```
dint DIntAt(void *Buffer, int Index);
dint DIntAt(int Index);
```

Parameters

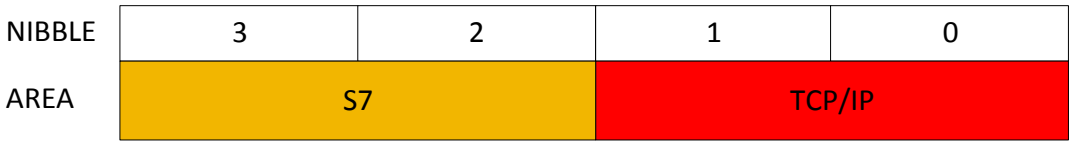
	Type	Dir.	
Buffer	Pointer	In	Pointer to an user buffer
Index	int	In	Start byte

Example

```
MyDInt=S7.DIntAt(&MyBuffer, 140);
```

Error codes

The result error code (a 16 bit value) is composed by two fields, as in figure:



Errors encoding

The first 8 bits (2 nibbles) represent a TCP/IP error. The last two nibbles represent an S7 protocol error

TCP Errors table

Mnemonic	HEX	Meaning
errTCPConnectionFailed	0x0001	TCP Connection error.
errTCPConnectionReset	0x0002	Connection reset by the peer.
errTCPDataRecvTout	0x0003	A timeout occurred waiting a reply.
errTCPDataSend	0x0004	Ethernet driver returned an error sending the data.
errTCPDataRecv	0x0005	Ethernet driver returned an error receiving the data.
errISOConnectionFailed	0x0006	ISO connection failed.
errISONegotiatingPDU	0x0007	ISO PDU negotiation failed.
errISOInvalidPDU	0x0008	Malformed PDU supplied.

S7 Errors table

Mnemonic	HEX	Meaning
errS7InvalidPDU	0x0100	Invalid PDU received.
errS7SendingPDU	0x0200	Error sending a PDU.
errS7DataRead	0x0300	Error during data read
errS7DataWrite	0x0400	Error during data write
errS7Function	0x0500	The PLC reported an error for this function.
errBufferTooSmall	0x0600	The buffer supplied is too small.

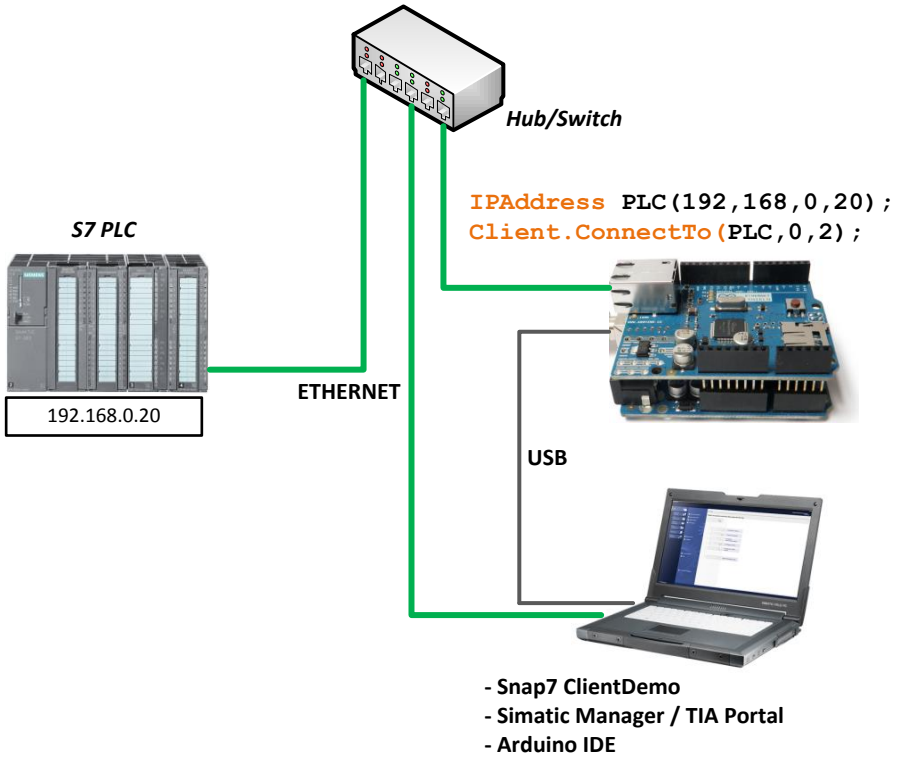
I made this division because in presence of a TCP error the Client should be disconnected and re-connected.

It's easy knowing this by masking the Hi-byte of the error code.

Testing Platforms

Let's see some typical configurations with that you can work.

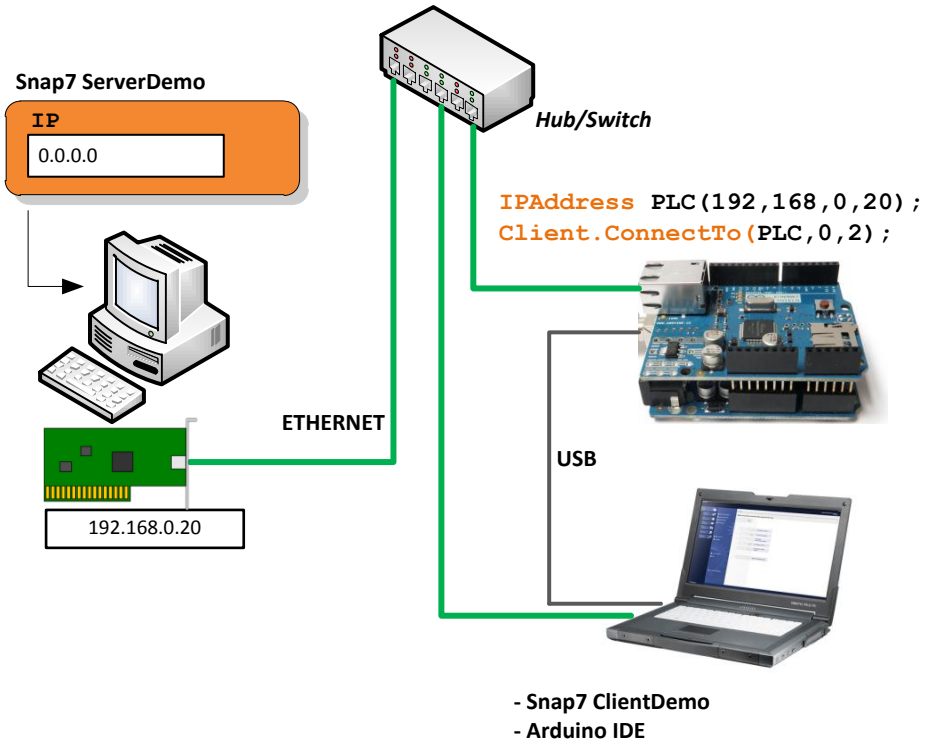
Luxury Platform



With this configuration you have the freedom to modify both PLC program/Data and Arduino Sketch.
This is a production configuration.

Snap7 ClientDemo is optional if you have Simatic Manager in the development PC.

Simulation/Debug Platform



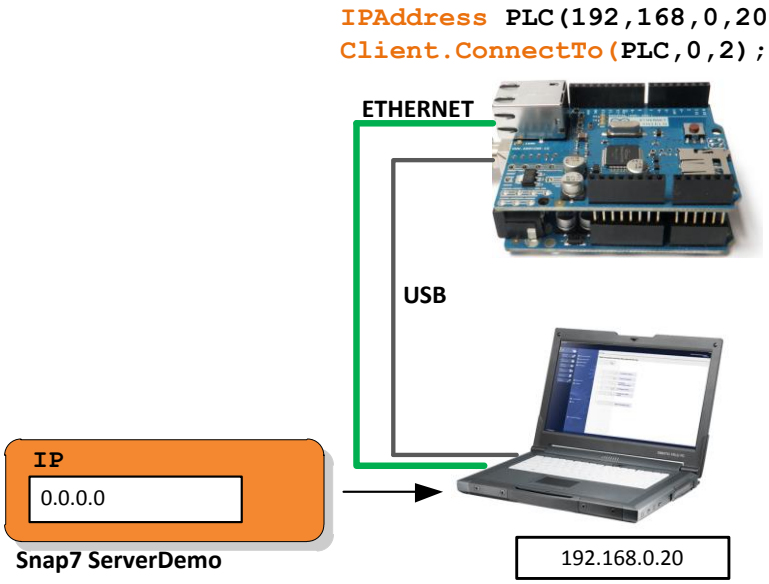
With this configuration you don't need a physical PLC to develop your sketches.

The IP = 0.0.0.0 in the Server window means that the Server is listening onto the default adapter (which in the example has IP = 192.168.0.20).

You don't need of Simatic Manager. Snap7Server offers a detailed log of the clients activities.

This configuration allows also the usage of Wireshark™ with S7Comm™ plugin for protocol debug.

Compact Platform



As of the above, here ServerDemo runs in the same development PC.

If you have a good resolution screen this configuration is very comfortable.

Also in this configuration you can use Wireshark + S7Comm.