Moka7

Reference manual

Davide Nardella

Rev.2 – December 15, 2016

Summary

Project overview

Moka7 is the Java 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 Java implementation of the S7Protocol.

Moka7 is deployed as a set of source code classes that you can use in your Java project to communicate with S7 PLCs.

Not all functions are ported but the list of the PLC managed is the same, it's designed to work with small hardware java-based, Android phones or even for large projects which don't needs of extended control functions.

Main features

- Fully standard Java code without any dependencies.
- Fully multiplatform, virtually every hardware with an Ethernet adapter able to run a JVM 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 convention.

Licensing

Moka7 is distributed as a source code library under dual license :

- **GNU Library** or **Lesser General Public License version 3.0 (LGPLv3)**
- **Eclipse Public License 1.0**

Basically this means that you have to choose in advance which take before you import the library into your project

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 MOKA7 PROJECT HAVE BEEN VIOLATED SOME COPYRIGHTS, PLEASE EMAIL US, AND ALL THE NECESSARY CHANGES WILL BE MADE.

About this manual

This manual describes only the Moka7 library, what it consists of and how to use it.

For a more detailed information about PLC interfacing and S7 Protocol please refer to the **Snap7 reference manual**.

Siemens S7 Protocol

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

Moka7, 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.

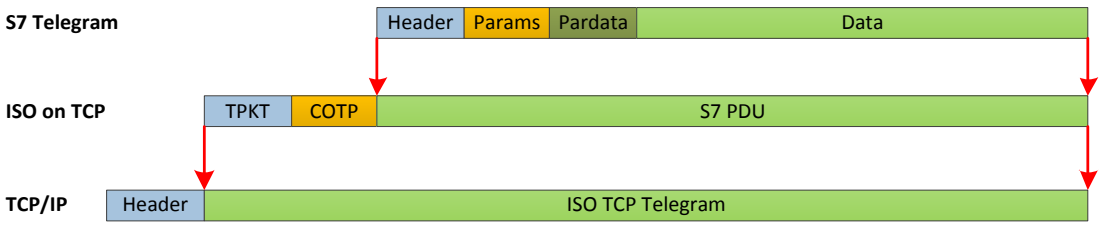
The first two elements are always present, the other are optional.

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 Moka7 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

S7 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/Set PLC Date and Time	**O**	**O**	**O**	**-**	**-**	**-**	**-**
Get PLC Status	**O**	**O**	**O**	**-**	**-**	**-**	**-**
Read SZL	**O**	**O**	**O**	**-**	**-**	**-**	**-**
Get AG Block Info	**O**	**O**	**O**	**-**	**-**	**-**	**-**
DB Get	**O**	**O**	**O**	**-**	**-**	**-**	**-**
Control Run/Stop	**O**	**O**	**O**	**-**	**-**	**-**	**-**
Security functions	**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, Moka7 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.

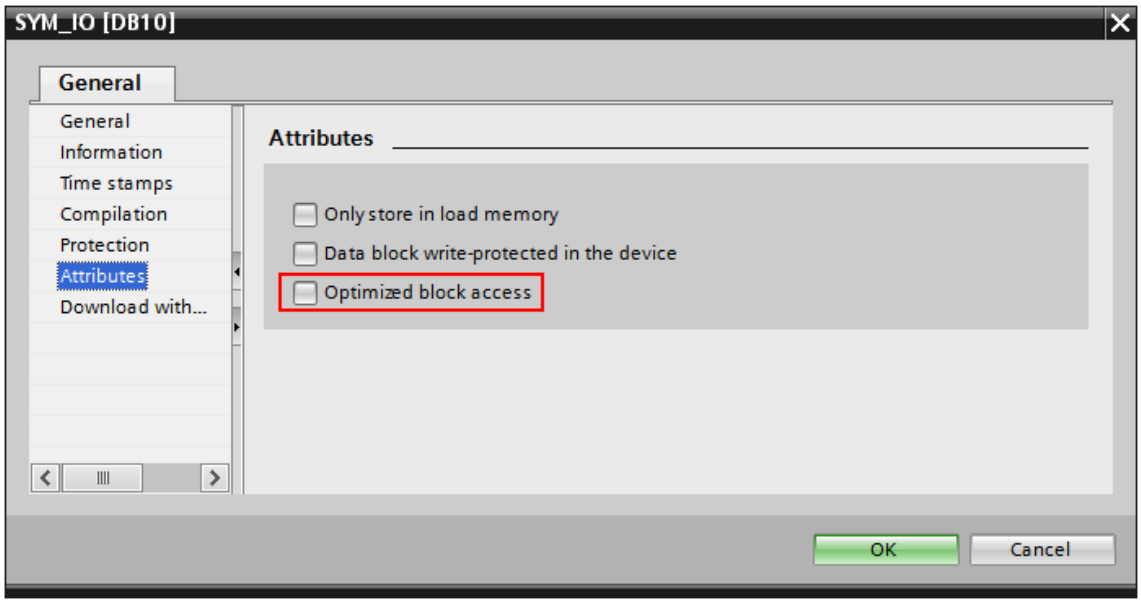
1. Only global DBs can be accessed.
2. The optimized block access must be turned off.
3. The access level must be "full" and the "connection mechanism" must allow GET/PUT.

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…")

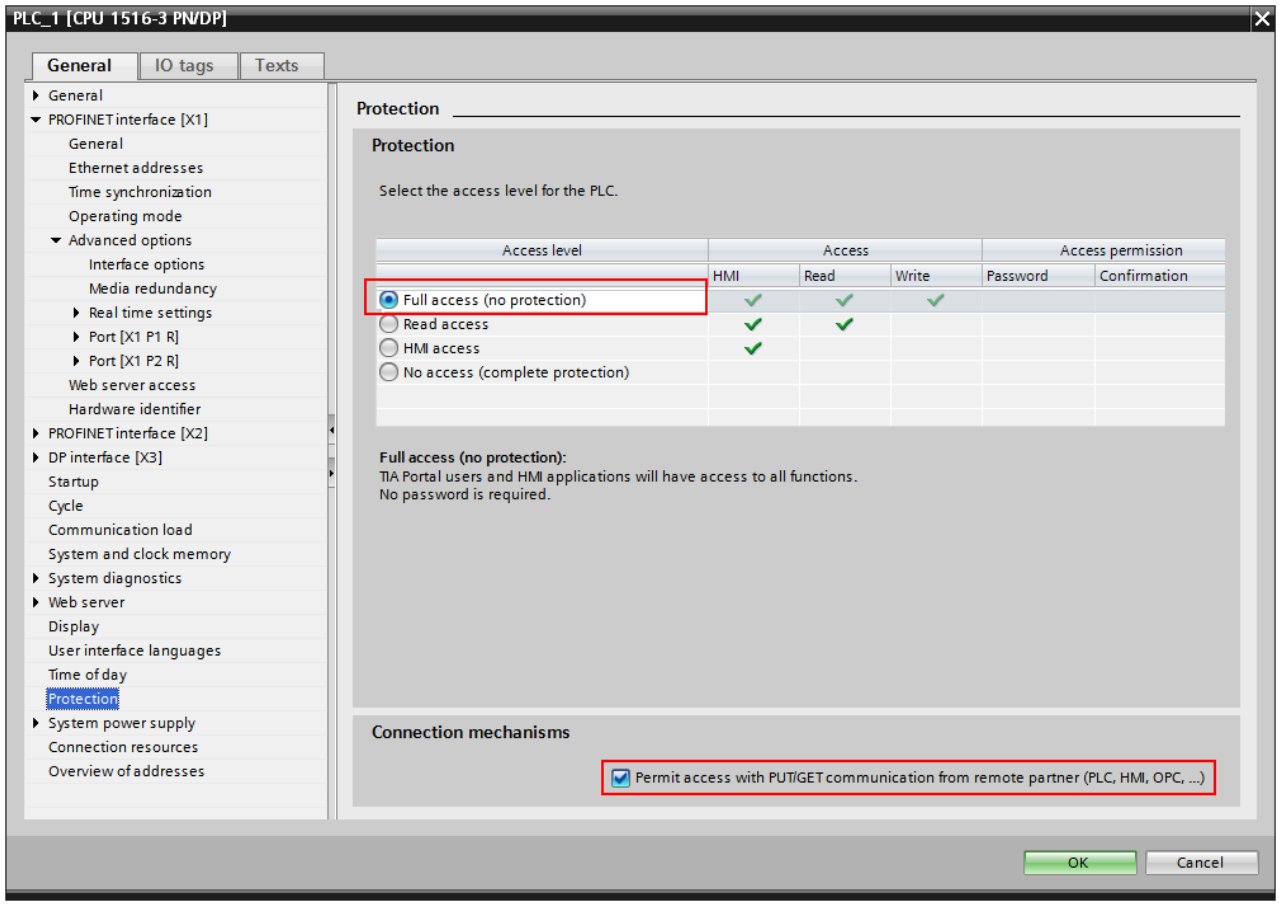
Uncheck Optimized block access, by default it's checked.



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 Moka7 point of view) S7 Protocol.

They are very different, and the first is not covered by Snap7 and Moka7 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.

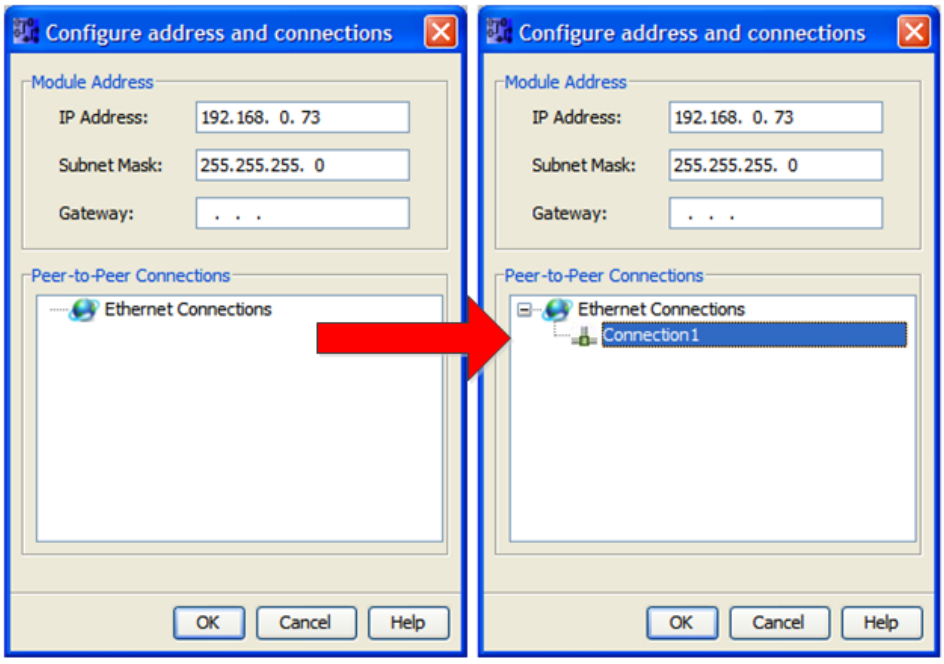
LOGO must be set as MASTER (i.e. NORMAL mode as LOGO Comfort says).

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

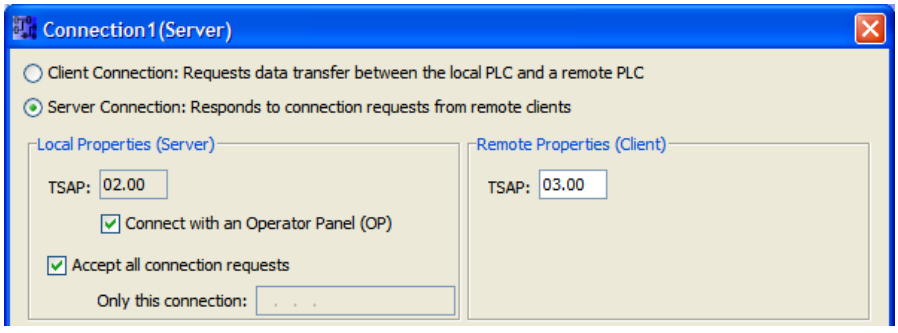
Connection configuration

Configuring a **server connection** allows you to connect Moka7 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**.

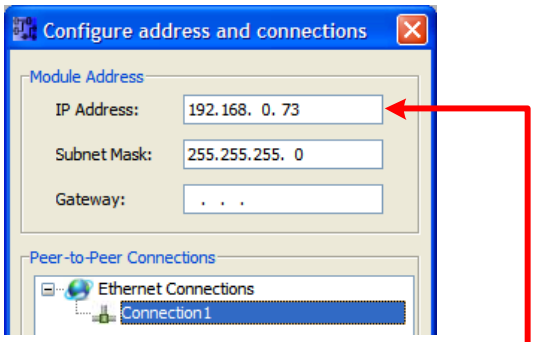


Note:

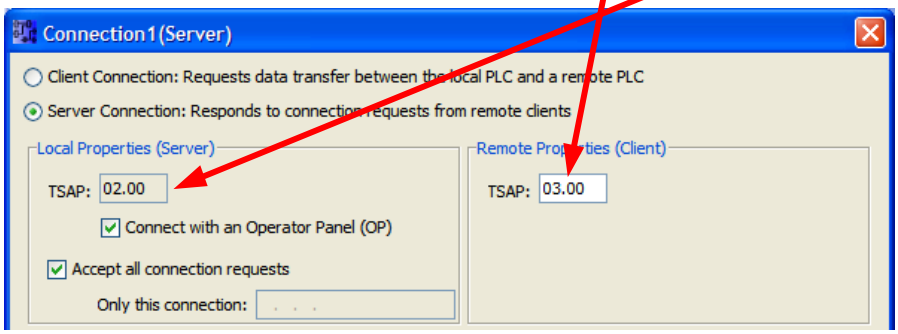
1. “Connect with an operator panel” checkbox can be checked or unchecked.
2. If you uncheck “Accept all connections” you must specify the PC address (for now I suggest you to leave it checked).

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.

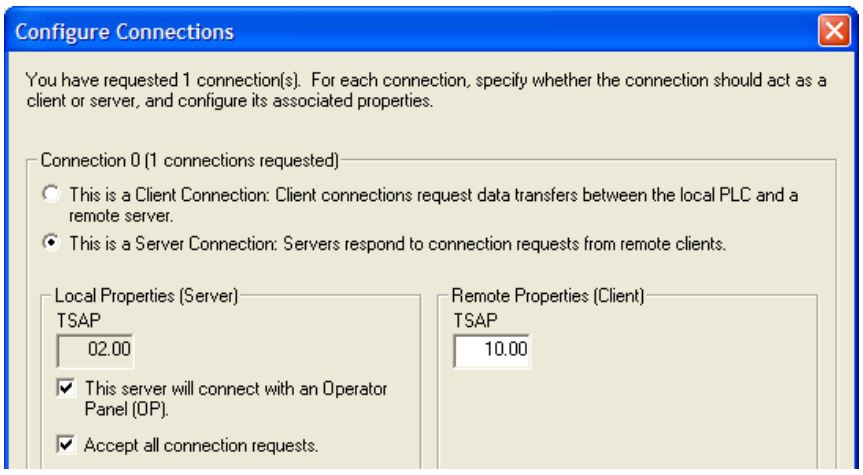
There are several tutorials in the Siemens site that show how to connect an HMI (via WinCC flexible or TIA) to the LOGO and the detailed map.
Please refer to them for further information.

S7 200 (via CP243)

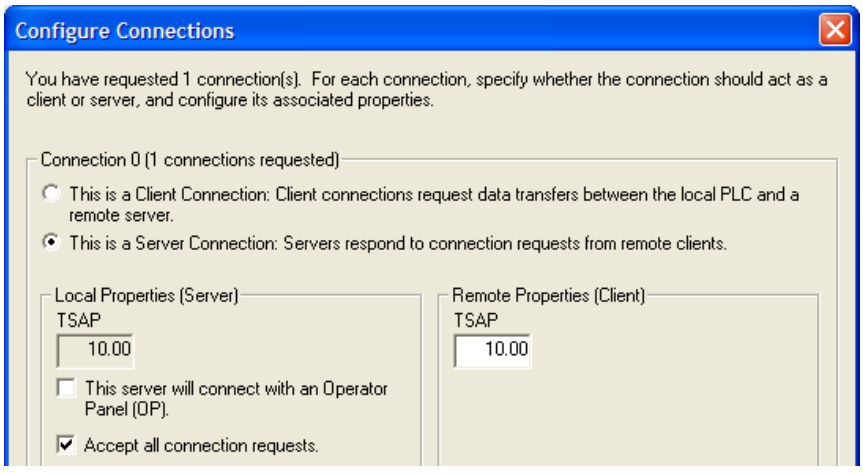
S7200 was out of the scope for Snap7 and Moka7 because beginning November 2013 the S7-200 product family entered into the Phase Out stage of its product life cycle, but after working with LOGO also this PLC can be accessed since it uses the same connection mechanism.

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.**

Moka7 deploy

Moka7 is a classes library, for convenience the source files are included into two projects both containing also a demo program.

A NetBeans 4.7 project called **Moka7-NetBeans**.

A Eclipse Kepler project called **Moka7-Eclipse.**

The projects are absolutely the same and contain **the same source files**, I made this to avoid 1000 email asking how to convert NetBeans project into an Eclipse project and vice-versa.

Into the project you will find two packages :

- **Moka7**, the main package.
- **Moka7Demo** that contains a Client demo program

Many files contained into Moka7 package are only structure classes, i.e. the java port of the C structures, the same that you find into snap7.h (see snap7 project):

- IntByRef.java (utility class to pass an integer by reference)
- S7BlockInfo.java
- S7CpInfo.java
- S7CpuInfo.java
- S7OrderCode.java
- S7Protection.java
- S7Szl.java

The main classes that we will analyze are:

- **S7Client**, the client object.
- **S7** an helper class to read/write S7 types from/to a byte array.

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.

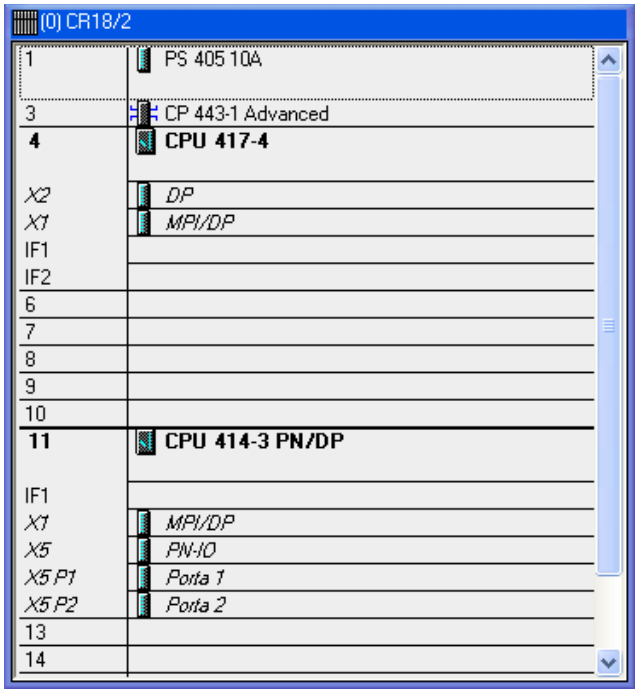
Rack and Slot

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.

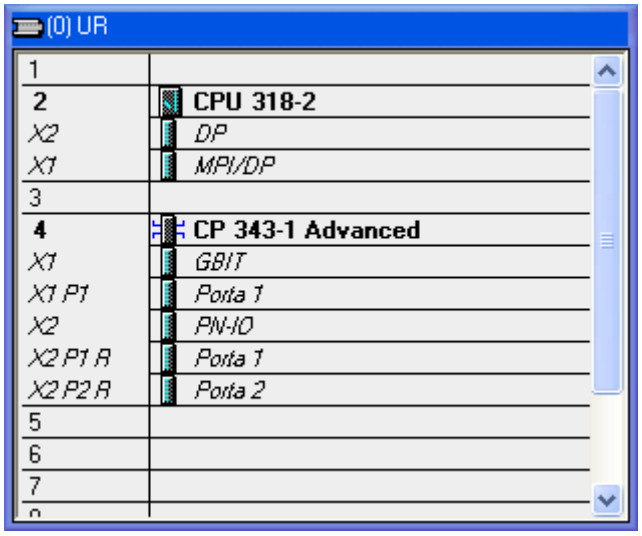
Let's see some examples of hardware configuration:



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

```
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.

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.

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, byte[] Data)
```

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 words to read (1)
Data	byte array	In	User buffer. (see remarks)

Area table

Helper Const	Value	Mean
S7.S7AreaPE	0x81	Process Inputs.
S7.S7AreaPA	0x82	Process Outputs.
S7.S7AreaMK	0x83	Merkers.
S7.S7AreaDB	0x84	DB
S7.S7AreaCT	0x1C	Counters.
S7.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.

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.

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

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 byte buffer into PLC.

Declaration

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

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

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

Block oriented functions

Function	Purpose
GetAgBlockInfo	Returns info about a given block in AG.
DBGet	Uploads a DB from AG.

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,
  S7BlockInfo Block)
```

Parameters

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

BlockType values

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

Return value

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

DBGet

Description

Uploads an entire DB from AG. As output *SizeRead.Value* will contain the size read.

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

Only data is uploaded (without the header).

Declaration

```
public int DBGet(int DBNumber, byte[] Data, IntByRef SizeRead)
```

Parameters

	Type	Dir.	
DBNumber	int	In	DB Number
Data	byte array	in	Address of the user buffer
SizeRead	IntByRef	In	IntByRef class instance

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.

This is an utility function implemented as above.

```
public int DBGet(int DBNumber, byte[] Buffer, IntByRef SizeRead)
{
    S7BlockInfo Block = new S7BlockInfo();
    // Query the DB Length
    LastError = GetAgBlockInfo(S7.Block_DB, DBNumber, Block);
    if (LastError==0)
    {
        int SizeToRead = Block.MC7Size();
        // Checks the room
        if (SizeToRead<=Buffer.length)
        {
            LastError=ReadArea(S7.S7AreaDB, DBNumber, 0, SizeToRead, Buffer);
            if (LastError==0)
                SizeRead.Value=SizeToRead;
        }
        else
            LastError=errS7BufferTooSmall;
    }
    return LastError;
}
```

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
Cli_GetPlcDateTime	Returns the PLC date/time.
Cli_SetPlcDateTime	Sets the PLC date/time with a given value.
Cli_SetPlcSystemDateTime	Sets the PLC date/time with the host (PC) date/time.

GetPlcDateTime

Description

Reads PLC date and time into a Java Date class instance.

Declaration

```
public int GetPlcDateTime(Date DateTime)
```

Parameters

	Type	Dir.	
DateTime	Date	In	Date 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(Date DateTime)
```

Parameters

	Type	Dir.	
DateTime	Date	In	Date 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, S7Szl SZL)
```

Parameters

	Type	Dir.	
ID	integer	In	List ID
Index	integer	In	List Index
SZL	S7Szl	in	S7Szl class instance

S7Szl fields (see S7Szl.java)

```
public int LENTHDR;
public int N_DR;
public int DataSize;
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

LENTHDR and N_DR are HI-LOW order swapped, the data buffer is unchanged.

GetOrderCode

Description

Gets CPU order code and version info.

Declaration

```
public int GetOrderCode(S7OrderCode Code)
```

Parameters

	Type	Dir.	
Code	S7OrderCode	in	S7OrderCode class instance

S7OrderCode fields and methods (see S7OrderCode.java)

```
public int V1;
public int V2;
public int V3;
public String Code()
```

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	in	S7CpuInfo class instance

S7CpuInfo fields (see S7CpuInfo.java)

```
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	in	S7CpInfo class instance

S7CpInfo fields (see S7CpInfo.java)

```
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.
GetPlcStatus	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.

GetPlcStatus

Description

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

Declaration

```
public int GetPlcStatus(IntByRef Status)
```

Parameters

	Type	Dir.	
Status	IntByRef	In	IntByRef class instance

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 trimmed, 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(S7Protection Protection)
```

Parameters

	Type	Dir.	
Protection	S7Protection class	in	S7Protection class instance

TS7Protection fields

```
public int sch_schal;
public int sch_par;
public int sch_rel;
public int bart_sch;
public int 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)

Return value

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

Miscellaneous functions

These are utility functions.

Function	Purpose
PduLength	Returns info about the PDU length (requested and negotiated).
ErrorText	Returns a textual explanation of a given error number.

PDULength

Description

Returns the value of the PDU length negotiated.

Declaration

```
public int PDULength()
```

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.

ErrorText

Description

Returns a textual explanation of a given error number.

Declaration

```
public static String ErrorText(int Error)
```

Parameters

	Type	Dir.	
Error	Integer	In	Error code

Remarks

The messages are in (internet) English.

Public Fields

RecvTimeout

int, **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.

S7 Helper syntax

Read functions

Function	Purpose
GetBitAt	Returns the Bit at given location in an user byte buffer
GetWordAt	Returns the Word at given location in an user byte buffer
GetShortAt	Returns the Short at given location in an user byte buffer
GetDWordAt	Returns the DWord at given location in an user byte buffer
GetDIntAt	Returns the DInt at given location in an user byte buffer
GetFloatAt	Returns the Float at given location in an user byte buffer
GetStringAt	Returns the String at given location in an user byte buffer
GetPrintableStringAt	Returns the String (in printable format) at given location in an user byte buffer
GetDateAt	Returns the Date and Time at given location in an user byte buffer

Write functions

Function	Purpose
SetBitAt	Writes a Bit at given location in an user byte buffer
SetWordAt	Writes a Word at given location in an user byte buffer
SetShortAt	Writes a Short at given location in an user byte buffer
SetDWordAt	Writes a DWord at given location in an user byte buffer
SetDIntAt	Writes a DInt at given location in an user byte buffer
SetFloatAt	Writes a Float at given location in an user byte buffer
SetDateAt	Writes a Date and Time at given location in an user byte buffer

GetBitAt

Description

Returns the Bit at given location of an user buffer.

Declaration

```
public static boolean GetBitAt(byte[] Buffer, int Pos, int Bit)
```

Parameters

	Type	Dir.	
Buffer	byte array	In	User buffer
Pos	int	In	Start byte
Bit	int	In	Bit number inside the byte

Example

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

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

SetBitAt

Description

Writes the Bit at given location into an user buffer.

Declaration

```
public static void SetBitAt(byte[] Buffer, int Pos, int Bit,
  boolean Value)
```

Parameters

	Type	Dir.	
Buffer	byte array	In	User buffer
Pos	int	In	Start byte
Bit	int	In	Bit number inside the byte
Value	boolean	In	Value to

Example

```
S7.GetBitAt(MyBuffer, 140, 2, true);
```

The third bit of the byte 140 of MyBuffer will be set to 1.

GetWordAt

Description

Returns the Word (16 bit unsigned integer) at given location of an user buffer.

Declaration

```
public static int GetWordAt(byte[] Buffer, int Pos)
```

Parameters

	Type	Dir.	
Buffer	byte array	In	User buffer
Pos	int	In	Start byte

Example

```
MyWord=S7.GetWordAt(MyBuffer, 140);
```

SetWordAt

Description

Writes the Word (16 bit unsigned integer) at given location into an user buffer.

Declaration

```
public static void SetWordAt(byte[] Buffer, int Pos, int Value)
```

Parameters

	Type	Dir.	
Buffer	byte array	In	User buffer
Pos	int	In	Start byte
Value	int	In	Value to write Range=0..65535

Example

```
S7.SetWordAt(MyBuffer, 140, 62340);
```

GetDWordAt

Description

Returns the DWord (32 bit unsigned integer) at given location of an user buffer.

Declaration

```
public static long GetDWordAt(byte[] Buffer, int Pos)
```

Parameters

	Type	Dir.	
Buffer	byte array	In	User buffer
Pos	int	In	Start byte

Example

```
MyDWord=S7.GetDWordAt(MyBuffer, 140);
```

SetDWordAt

Description

Writes the DWord (32 bit unsigned integer) at given location into an user buffer.

Declaration

```
public static void SetDWordAt(byte[] Buffer, int Pos, long Value)
```

Parameters

	Type	Dir.	
Buffer	byte array	In	User buffer
Pos	int	In	Start byte
Value	long	In	Value to write Range=0.. 4294967295

Example

```
S7.SetDWordAt(MyBuffer, 140, 200000);
```

GetFloatAt

Description

Returns the Float (32 bit floating point number) at given location from an user buffer.

Declaration

```
public static float GetFloatAt(byte[] Buffer, int Pos)
```

Parameters

	Type	Dir.	
Buffer	byte array	In	User buffer
Pos	int	In	Start byte

Example

```
MyFloat=S7.GetFloatAt(MyBuffer, 140);
```

SetFloatAt

Description

Writes the Float (32 bit floating point number) at given location into an user buffer.

Declaration

```
public static void SetFloatAt(byte[] Buffer, int Pos, float Value)
```

Parameters

	Type	Dir.	
Buffer	byte array	In	User buffer
Pos	int	In	Start byte
Value	float	In	Floating point value to write

Example

```
S7.SetFloatAt(MyBuffer, 140, -2.56);
```

GetShortAt

Description

Returns the Short (16 bit signed integer) at given location of an user buffer.

Declaration

```
public static int GetShortAt(byte[] Buffer, int Pos)
```

Parameters

	Type	Dir.	
Buffer	byte array	In	User buffer
Pos	int	In	Start byte

Example

```
MyShort=S7.GetShortAt(MyBuffer, 140);
```

SetShortAt

Description

Writes the Short (16 bit signed integer) at given location into an user buffer.

Declaration

```
public static void SetShortAt(byte[] Buffer, int Pos, int Value)
```

Parameters

	Type	Dir.	
Buffer	byte array	In	User buffer
Pos	int	In	Start byte
Value	int	In	Value to write Range=-32768..32767

Example

```
S7.SetShortAt(MyBuffer, 140, -15000);
```

GetDIntAt

Description

Returns the Double Integer (32 bit signed integer) at given location of an user buffer.

Declaration

```
public static int GetDIntAt(byte[] Buffer, int Pos)
```

Parameters

	Type	Dir.	
Buffer	byte array	In	User buffer
Pos	int	In	Start byte

Example

```
MyDInt=S7.GetDIntAt(MyBuffer, 140);
```

SetDIntAt

Description

Writes the Double Integer (32 bit signed integer) at given location into an user buffer.

Declaration

```
public static void SetDIntAt(byte[] Buffer, int Pos, int Value)
```

Parameters

	Type	Dir.	
Buffer	byte array	In	User buffer
Pos	int	In	Start byte
Value	int	In	32 bit Integer value to write

Example

```
S7.SetDintAt(MyBuffer, 140, 200000);
```

GetDateAt

Description

Returns the S7 Date and Time value at given location of an user buffer.

Declaration

```
public static Date GetDateAt(byte[] Buffer, int Pos)
```

Parameters

	Type	Dir.	
Buffer	byte array	In	User buffer
Pos	int	In	Start byte

SetDateAt

Description

Writes the S7 date and time value at given location into an user buffer.

Declaration

```
public static void SetDateAt(byte[] Buffer, int Pos, Date DateTime)
```

Parameters

	Type	Dir.	
Buffer	byte array	In	User buffer
Pos	int	In	Start byte
DateTime	Date	In	Calendar class instance

GetStringAt

Description

Returns the String (UTF-8 char array) at given location of an user buffer.

Declaration

```
public static String GetStringAt(byte[] Buffer, int Pos, int MaxLen)
```

Parameters

	Type	Dir.	
Buffer	byte array	In	User buffer
Pos	int	In	Start byte
MaxLen	int	In	Number of chars expected

GetPrintableStringAt

Description

Returns the String (UTF-8 char array) at given location of an user buffer, not printable chars into the string are replaced by '.'

This function is useful for dump tasks.

Declaration

```
public static String GetPrintableStringAt(byte[] Buffer, int Pos,
   int MaxLen)
```

Parameters

	Type	Dir.	
Buffer	byte array	In	User buffer
Pos	int	In	Start byte
MaxLen	int	In	Number of chars expected

Error codes

Mnemonic	HEX	Meaning
errTCPConnectionFailed	0x0001	TCP Connection error.
errTCPDataSend	0x0002	TCP error sending the data.
errTCPDataRecv	0x0003	TCP error receiving the data.
errTCPDataRecvTout	0x0004	A timeout occurred waiting a reply.
errTCPConnectionReset	0x0005	Connection reset by the peer.
errISOInvalidPDU	0x0006	Malformed PDU supplied.
errISOConnectionFailed	0x0007	ISO connection failed.
errISONegotiatingPDU	0x0008	ISO PDU negotiation failed.
errS7InvalidPDU	0x0009	Invalid PDU received.
errS7DataRead	0x000A	Error during data read.
errS7DataWrite	0x000B	Error during data write.
errS7FunctionError	0x000D	The PLC reported an error for this function.
errBufferTooSmall	0x000C	The buffer supplied is too small.
errInvalidParams	0x000E	An invalid parameter was supplied to the function.