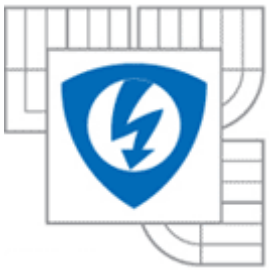
UAMT FEI



Remote management system for PLC

Diploma Thesis

Daniel Luis Bolaños Conde

23/05/2011

INDEX

1.INTRODUCTION

A PLC is an electronic element which can be found in all the industrial environment. Since they appeared, about in 1969 in United States until now, they have been becoming important , and now they are one essential element in every system.

Due to the fact that they can be found everywhere, science and technology have made advances so they can be controlled or monitored in the easiest possible way. Then, the aim is to locate some manner to communicate them, so it could be possible to carry out maintenance operations too, for example.

Nowadays, thanks to the advances of Ethernet and Internet it has been achieved that by means of different communication protocols, the communication between PLC's or between PLC's and PC's is possible in such a way that one device could be in China and the other in USA.

Below are shown some illustrations:

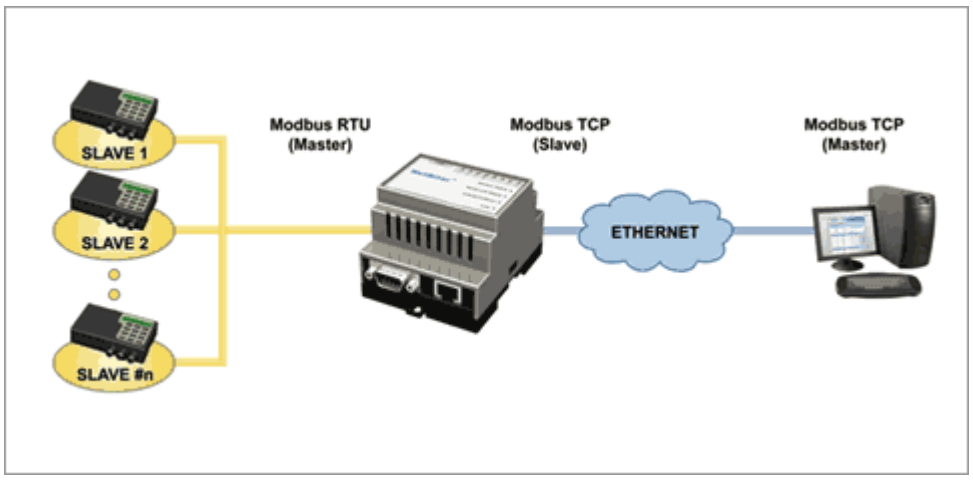


Figure 1. Conversion from Modbus/TCP to Modbus RTU

In this first figure it can be observed the connection between Modbus RTU products to SCADA or PLC's over Ethernet (There is a conversor between Modbus RTU and ModbusTCP).

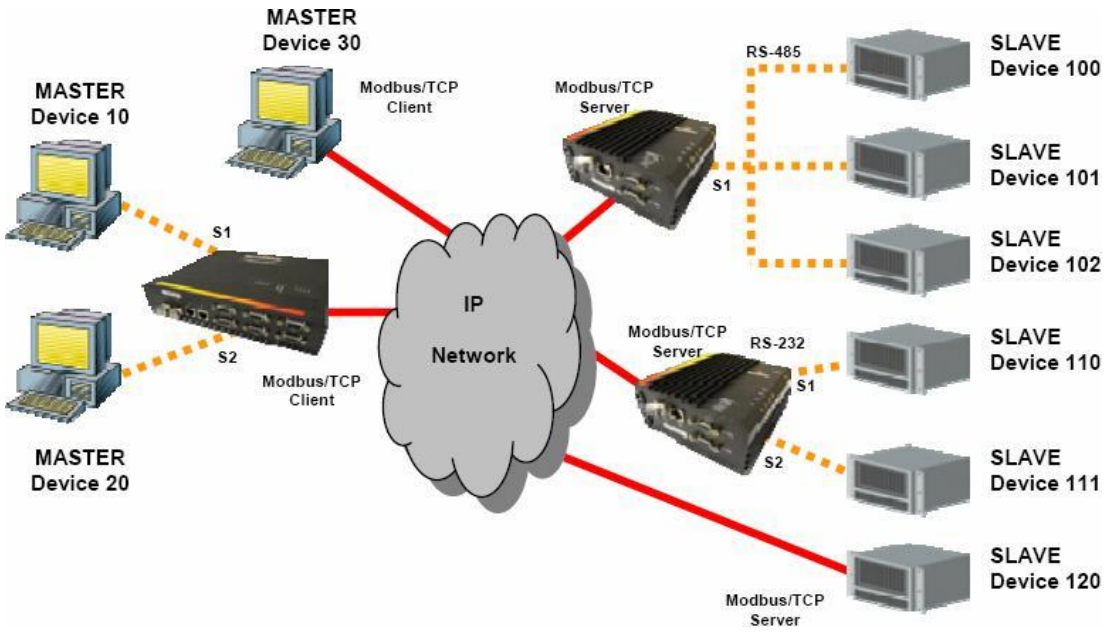


Figure 2. Typical Modbus/TCP structure

Something similar is shown in this second illustration.

2.- OBJECTIVE

The goal of this thesis is to propose and implement a library of PLC-side function blocks and a PC application used for remote management of PLCs.

Tasks:

1.- Prepare a detail concept of the remote management system.

2.- Implement a library of function blocks for safe PLC-side data archiving.

3.- Implement functions for archived data retrieval and saving to USB drive.

4.- Implement PLC functions and a PC application for transfering the archived data from PLC to PC using a selected TCP based protocol.

5.- Carry out testing and create a detailed documentation of both the PLC and PC software and libraries, as well as the communication protocol used.

6.- Prepare a brief end-user manual for both the PLC functions library as well as the PC application.

3.- PLC USED AND AUTOMATION STUDIO[1]

The PLC used for achiving the objectives is one from the brand BR in the SG4 family, its main features are the following:

Description:
Power Panel PP45
5,7" QVGA monochrome LC Display
with touch screen (resistive),
10 touch-keys,
64 MB DRAM, 48 KB SRAM
Compact Flash Slot,
ETH 10/100, X2X Link, 2x USB,
IP65 protection (from front),
order application memory separately.
Order TB103 and TB704 terminal blocks
separately.

Figure 3. Front image of the PLC

B&R Automation Studio is the integrated software development environment that contains tools for all phases of a project. The controller, drive, communication and visualization can all be configured in one environment. That reduces both integration time and maintenance costs.

The main features of Automation Studio are:

- Project management - Software on demand. Thanks to the system-oriented view of the project and division into functional packages, extensive projects can be clearly managed and programmed.
- Programming - The right programming language for every application. The user is provided effective support for the programming by the languages integrated in Automation Studio (IEC 61131-3 and ANSI C).

[1] Almost all the information to create this section has been obtained from the official website of B&R.[9]

- Integrated visualization.
- Configuring drives - Simple solutions for positioning tasks.
- Diagnostics and simulation. Automation Studio provides a wide selection of diagnostic tools for reading system information and for optimizing the system. (Logger, Debugger, line coverage, profiler....)
- Remote maintenance - Consistent from process to firmware exchange.
- Communication and fieldbus systems. Real-time operating systems - Scalability and investment security. An integral component of Automation Studio is the real-time operating system, the software kernel that allows applications to run on a target system. This guarantees the highest possible performance for the hardware being used.

4.- SELECTION OF THE LANGUAGE USED[2]

As it was mentioned before, there exist some programming languages which can be used in Automation Studio, a comparison between them is the following:

	LAD	FBD	CFC	SFC	IL	ST	AB	C
Logic	√	√	√	√	√	√	√	√
Arithmetic					√	√	√	√
Decisions	√	√	√	√	√	√	√	√
Loops						√	√	√
Step sequencers				√		√	√	√
Dyn. variables						(√)	√	√
Function blocks	√	√	√	√	√	√	√	√

Table 1. Comparison of possibilites of the different languages

As it can be observed, the three last programming languages, Structured Text (ST), Automation Basic (AB), and C, allow all the features to create programs. Finally, the decision is to use Structure Text and the main reason is because is one of the programming languages of the IEC 61131-3 standard (standardization languages of industrial control). A brief description and its main features are shown below.

Structured Text is a high level language. For those who are comfortable programming in Basic, PASCAL or Ansi C, learning Structured Text is simple.

[2] The information for this section has been created from the Training module *Automation Studio Basis*, which has the basic information.[5]

Structured Text (ST) has standard constructs that are easy to understand, and is a fast and efficient way to program in the automation industry.

The main characteristics are the following:

- High-level text language
- Structured programming
- Easy to use standard constructs
- Fast and efficient programming
- Self explanatory and flexible use
- Similar to PASCAL
- Easy to use for people with experience in PC programming languages
- Conforms to the IEC 61131-3 standard

```
PROGRAM _INIT

  (* calculate max. number of stations *)
  ModGrpNrMax:= UDINT_TO_UINT(SIZEOF(Station)/SIZEOF(Station[0]));
  (* calculate max. number of modules per station *)
  ModNrMax:= UDINT_TO_UINT(SIZEOF(Station[0].Modul)/SIZEOF(Station[0].Modul[0]));

  ModGrpNrError:= READY_FOR_NEW_ENTRY;            (* reset module group number for errorhandling *)
  ModNrError:= READY_FOR_NEW_ENTRY;               (* reset module number for errorhandling *)

END_PROGRAM

PROGRAM _CYCLIC

  tmpModOK:= TRUE;
  FOR ModGrpNr:= 0 TO (ModGrpNrMax - 1) DO
    FOR ModNr:= 0 TO (ModNrMax - 1) DO
      (* check only configured modules *)
      IF (Station[ModGrpNr].Modul[ModNr].Used = TRUE) THEN
        (* check only those configured modules with a connected PV on ModulesOK-Flag *)
        IF (Station[ModGrpNr].Modul[ModNr].pModulOkPv <> 0) THEN
          dModulOK ACCESS Station[ModGrpNr].Modul[ModNr].pModulOkPv;
          Station[ModGrpNr].Modul[ModNr].OK:= dModulOK;
          IF (Station[ModGrpNr].Modul[ModNr].OK = FALSE) THEN
            tmpModOK:= FALSE;                     (* set error *)
            (* error not recognized yet and logger ready for a new entry *)
            IF (BIT_TST(Station[ModGrpNr].Modul[ModNr].StatusInfo, 0) = FALSE) AND (ModGrpNrError = READY_FOR_NEW_ENTRY) THEN
              Station[ModGrpNr].Modul[ModNr].StatusInfo:= BIT_SET(Station[ModGrpNr].Modul[ModNr].StatusInfo, 0);
              ModGrpNrError:= ModGrpNr;           (* set module group number for errorhandling *)
              ModNrError:= ModNr;                 (* set module number for errorhandling *)
            END_IF;
          ELSE
            Station[ModGrpNr].Modul[ModNr].StatusInfo:= BIT_CLR(Station[ModGrpNr].Modul[ModNr].StatusInfo, 0);
          END_IF;
        ELSE
          (* no PV connected on ModuleOK-flag *)
          Station[ModGrpNr].Modul[ModNr].OK:= FALSE;
        END_IF;
      ELSE
        (* module not configured *)
        Station[ModGrpNr].Modul[ModNr].OK:= FALSE;
      END_IF;
    END_FOR;
  END_FOR;
  AllModulesOK:= tmpModOK;

END_PROGRAM
```

Figure 4. Example of a program in Structured Text.

5.- DATA STORAGE[3]

As the name suggests, data storage refers to saving data in memory on the controller (mostly ROM and mass memory, but also RAM). This can include data such as machine configuration, recipe data, operating hours, etc. The user must choose how and on which target memory the data should be saved. In most cases, nonvolatile (the contents are not lost even after a power failure) memory is used for this. The following possibilities are available for long-term data storage on B&R controllers:

- B&R data objects
- Files
- Variables (remanent and permanent)

Data objects consist of a header, the actual data and a checksum

Figure 5. Structure of a Data Object

The checksum on all B&R objects (including tasks and system objects) is monitored cyclically while the system is running to detect and react to errors such as unauthorized access with pointers. The monitoring is carried out in the idle time. 512

[3] The information of this chapter is from the training module Memory management and Data storage, except the conclusion.[6]

bytes per system tick are checked. Therefore, it can take up to a few minutes (depending on the size of the application) before the checksum monitor responds in the case of an error.

Data objects are the safest way to save data because a backup copy is automatically made as part of the application. Therefore, if a data object is destroyed during a write procedure (e.g. in the event of a power failure), a backup copy of the data object is still available and is automatically recovered when the controller is restarted.

On SG4 (The latest systems) systems it is possible to store files in mass memory on the controller. Mass memory includes hard drives (HDD), floppy disk drives (FDD), Compact Flash cards or USB storage media. The data is organized as on a PC and saved on logical drives (also in folders if desired).

Files can also be edited using a PC and transferred back to the controller when needed. This is a big advantage when e.g. recipes need to be created once and then transferred to other identical machines.

The following options are available for doing this:

- Removing the Compact Flash and connecting to a PC using a suitable adapter.
- Copying the files from the Compact Flash card to USB storage using the "FileIO" library.
- Access to the Compact Flash card via Ethernet FTP.
- The PLC copies the Data to a FTP server.

Using files makes it possible to easily manipulate and exchange data between controllers and PCs.

Finally, here is a summary table about files and data objects:

	Files	Data objects
Memory	Mass Memory	UserROM, SystemROM, UserRAM, DRAM
Access	Function blocks and dynamic variables	Function blocks and dynamic variables
Controller system	SG4	SG3, SG4
Monitoring, Checksum	No	Yes
Library	FileIO	DataObj
Can be transfered to PC	FTP, Compact Flash, USB	No
Can be edited on PC	Any program	Automation Studio

Table 2. Comparison between files and data objects.

It can be concluded that to save the archive using data objects is the safest way. However, attention should be paid to the fact that using large data objects is critical due to the time consumption. So it has to be found a way to create not so large data objects.

The files will also be used, when a phisycal person is next to the PLC and he wants to save the archive in a USB stick, as soon as the USB is inserted, the archive is saved into the stick as a CSV file.

Remanent variables could be also interesting to create one so in case of a power loss, the archiving process could start at the same point before the power loss.

6.-PROPOSED ARCHITECTURE

First of all, a general description of the library is given:

The library is installed in the PLC, this PLC will be located somewhere controlling/monitoring a process, for example it could be something related with the control of temperatures or pressures.

Those values of the inputs, are saved in the memory of the PLC. A group of objects have to be created, because they should not be really large, due to the fact that working with large data objects is bad for the time consumption.

The programmer of the PLC will be able to specify how much memory he wants to use to save the archive, and the maximum length of the objects where the records will be. Using the created library the system will know how many objects to create and their length. Due to the fact that generally the space will not be exact to create a number of data objects, it will exist one last smaller data object.

The structure of one record is shown as follows:

Date&Time	Real values	Boolean values

Figure 6. Structure of a Record

The first member will always be a variable of type Date&Time, which will be used to know when the record was saved in the memory. After that, the group of real values and finally the group of boolean values.

A numeric example of a system archiving 5 reals and 5 boolean values using this library in an archiving size of 700 bytes and a maximum size of 120 bytes per object is given below.

First of all, the size of one record is calculated, 5 real values (4*5=20 bytes) and 4 boolean values (1*5=5 bytes), besides of the date and time variable (4 bytes), it is a total of 29 bytes per record.

So if there is 120 bytes per object and each record occupies 29, there will be 4 records per object, and each object will be of a size of 116 bytes.

Due to the fact that the maximum size for archiving is 700 bytes and 116 bytes is the size of each object, there will be 6 objects, it is a total size of 696 bytes.

Because of rest of the space is only 4 bytes, there will not be a smaller object (no enough space for only one record in that object).

So, the main propose of the library is the creation of the appropriate number of data objects, and fill them with the records.

Besides this, in the library there are some function blocks to obtain for example some specific records or to get the name of the variables.

The library, called *Remote*, which will be responsible for solving this problem of automatical archiving, save the archive in a CSV file a USB stick and the communication with a PC consists of the next components:

- A datatype called *Control* which will contain some important parameters for the archiving. One of these will have to be created by the programmer as a global variable.
- Two function blocks known as *ArchReal* and *ArchBool*, which will carry out three functions, counting the number of connected inputs, save both the addresses of those inputs and their names in the memory, and finally copy the name in a global variable so it can be compared later. For each

variable that the programmer wants to archive, he needs to use once these blocks, if it is a real variable, *ArchReal*, if it is a boolean value, *ArchBool*.

- *Engine*, which we could say it is the main function block of the system, due to the fact that is the responsible for the creation of the data objects and save the records into them. It will be a must, to use this function block in all the archiving systems.
- *CreateArrays*, another function block, and its purpose is to create dynamic arrays when they are required.
- A function block called *DObjCreation* that is used inside *Engine* when it is needed to create the Data Objects.
- *DObjWriting*, which is a function block and it will be used the writing process in the data objects is going to start.
- Another block called *ArchInfo* that will be useful to obtain some parameters such as number of records, size of one record...
- A function block known as *RecRead* used to read a specific record of the archiving system.
- *Online_monitor*, which is used to connect variables that will be available to monitor online in the PC-side application.
- *Server_create*, a function block to create a server with the PLC.
- *ArchivetoCSV*, this function block will be used to save the archive in a CSV format file in a USB stick connected to the PLC.
- A function block called *ReadArchive* to read all the values present in the archive, starting from the oldest one.

Now theses components will be explained in more detail:

Data Type *Control*

It is a structure that consists of the following components:

- AdrREAL. It is an UDINT (A 32 bit number in the unsigned double integer format) which „points“ to the address of the first

real input of the system. It is the first element of an array which has the addresses of all the real inputs.

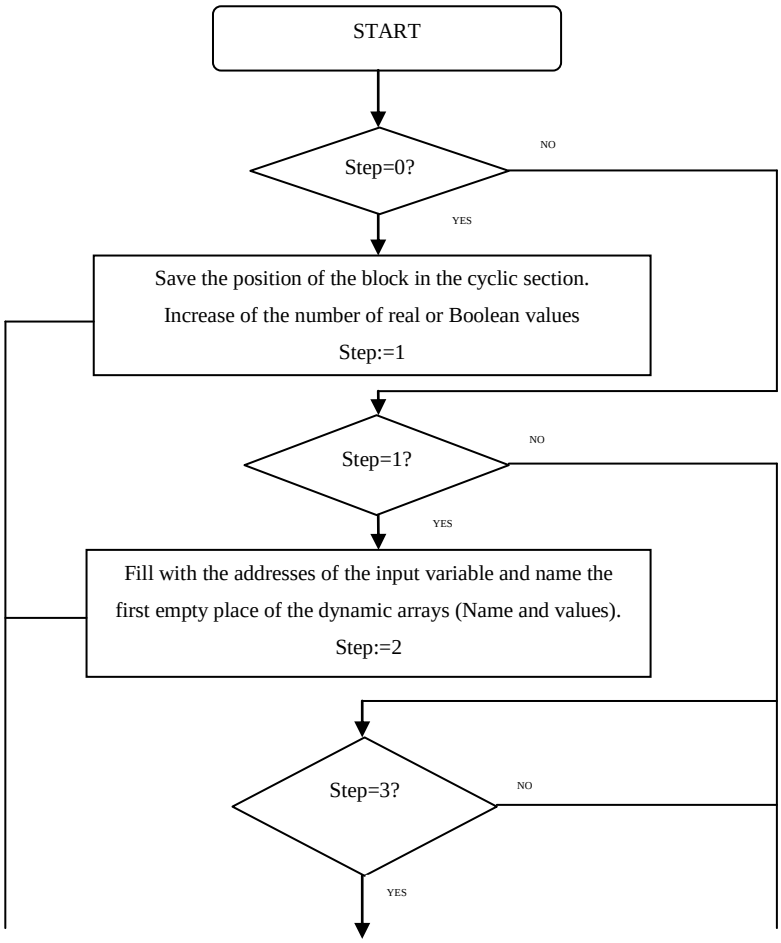
- AdrBOOL. The same but for the bolean inputs.
- NumberofR. An UINT (A 16 bit number in the unsigned integer format) variable which contents the number of connected real inputs.
- NumberofB. The same for the boolean variables.
- AdrDObjID. It will be used to store the address where the ID‘s of the Data Objects created can be found. It is the first element of an array which has the ID’s of the different Data Objects created.
- Offset. It has the position (offset) in the object which is currently being writing, where the next record has to be saved.
- Overwrite. A Boolean variable used to take into account when the process of overwriting has started.
- Recordstowrite. It has the number of records that are still possible to write in the current object.
- dwIdent. This variable has the ID of the current object, the one which is currently being written.
- CurrentDObj. It has the number of the data object in use.
- NumberDObj. Here it can be found the number data objects created for the archiving process.
- RecordsperObject. The numbers of records which can be placed in a data object.
- AdrNameReal. It is an UDINT (A 32 bit number in the unsigned double integer format) which „points“ to the address of the name of the first real input of the system. It is the first element of an array which has the addresses of all the real names.
- AdrNameBool. The same for the Boolean inputs.
- PeriodforArchiving. This variable will store the time after a cycle of archiving is carried out.

- NumberofRecords. Here it is possible to find the number of records contained in the whole archive.
- RecordSize. The size of one record.
- ArchStarted. Boolean variable so when the writing process is started is activated.
- SizelastDObj. The size of the last smaller data object
- RecordsLastObject. The number of records that it is possible to write in the last data object.
- IdentLR. A variable to know the ident of the object where the last record has been written.
- OffsetLR. A variable to know where in the object has been written the last record.
- NameInput. A variable to store the name of the inputs and compare it with another required name.
- Positionof. It is used to save the position in the cyclic program of a function block.
- Locked. Boolean variable used to block the archiving process in case the whole archive is being transfered to the PC or the PLC is creating a CSV file in a USB stick.
- NReals. An INT (A 16 bit number in the signed integer format) for the number of reals.
- NBooleans. Number of booleans to monitor.
- AdrNameReal1. Address of the first element of the array for the name of the reals to be monitor.
- AdrNameBool1. Address of the first element of the array for the names of the booleans to be monitor.
- AdrValReal. Address of the first element of the array for the real values to monitor.
- AdrValBool. Address of the first element of the array for the boolean values to monitor.

Function Blocks *ArchRealValue()* and *ArchBoolValue()*

These blocks have three inputs, one of type Control, one for the real or boleean input (the value which is going to be archived), ant the third for the name of the variable connected.

These two function blocks are almost equal, so they will be described together. A flowchart describing the operation is below (Fig. 6):



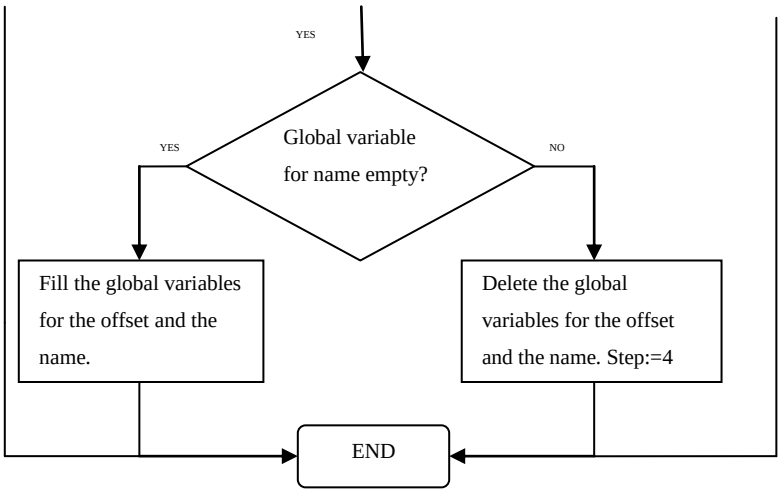


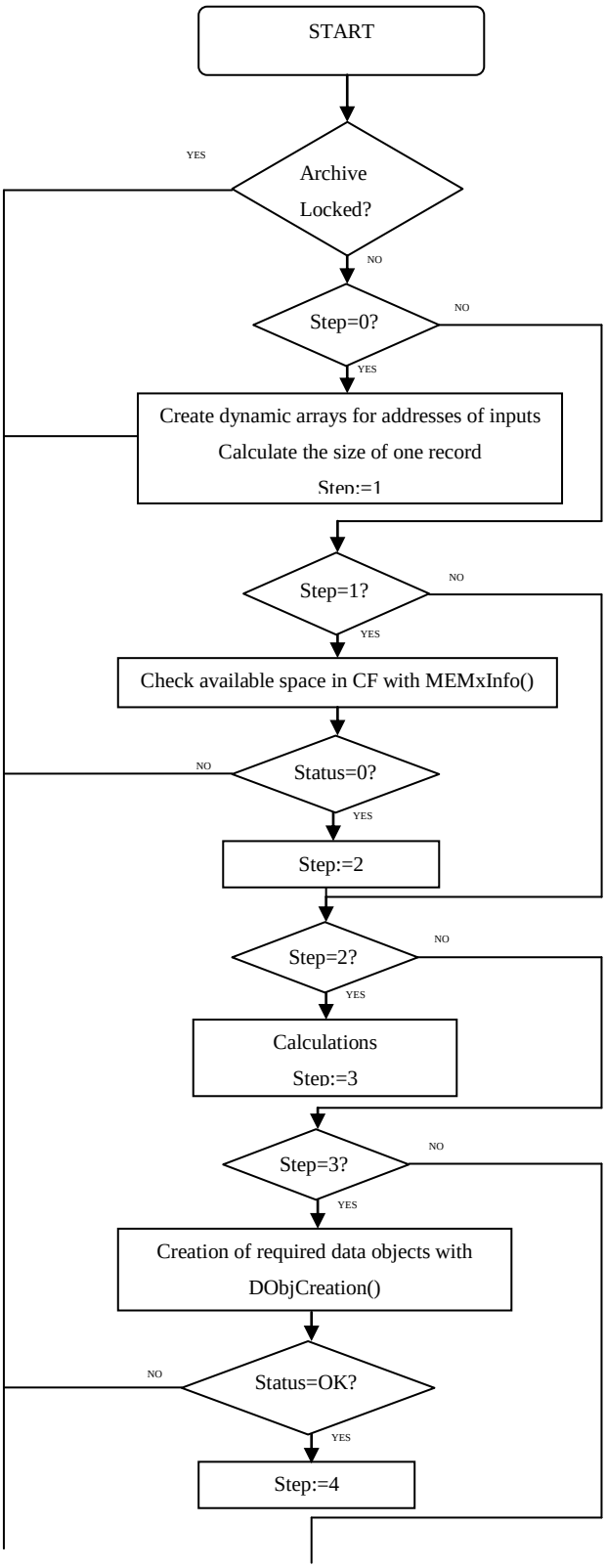
Figure 7. Flowchart of ArchReal() or ArchBool()

As seen, the function block consists of three steps, in the first is saved the position where the programmer has placed the block in the cyclic program (According to the value of the variable with the number of real or boolean inputs until this moment), and the variable of the structure type control (see previous paragraph) is increased one unit. Finally, step is set to value 1.

In the second step (step=1), two dynamic arrays that will be created in other function block using the number of real and boolean variables calculated in the first step, are filled with the addresses of the inputs and the addresses of the names. Then step is fixed to 3.

In the final step, if the globlal variable used in other blocks to compare the name of the inputs is empty, it will be filled with the name of the input of the current block, in the same way, the position of this block is saved in other global variable. In case this global variable is not empty, it is cleared so other functions can use it.

Function Block *Engine()*



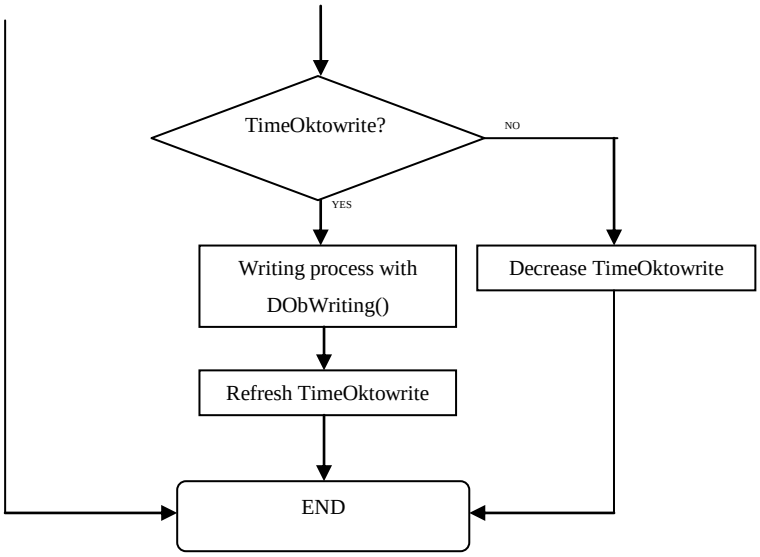


Figure 8. Flowchart of Engine()

This block has five inputs, one for a Control type variable, one for the desired space used for achiving, the third will be used to specify the lenght of the smaller data objects. The two last inputs are to set the values of the period for archiving and the cycle time.

In this function block can be observed several differentiated steps. In the first (Step=0), the dynamic arrays to store the addresses of the inputs and the addresses of the names will be created, also the size of one record will be calculated. After this step is used the return instruction, this is because in the following steps it will be needed that the dynamic arrays are filled.

The second step, when the value of Step is 1, the space available in the compact flash inserted in the PLC will be checked. Due to the function block *MEMxInfo()* is executed asynchronous, this step will be repeated until the output *Status* is set to 0.

After this, some important parameters, these are the number of objects, the number of records per object (also for the last smaller object), the size of the data

objects, a variable to control the number of cycles after which a archiving cycle has to take place. In addition, another dynamic array for storing the ID's of the data objects will be created.

In the next step (Step=3), the required data objects will be created with DobjCreation (in more detail in a later section). Because of the same reason as *MEMxInfo()*, it will be executed until the value of the status is OK.

Finally, when (Step = 4) the data is written in the data objects, is executed cyclicly. At the begining is checked if the number of cycles happened are enough to start a writing process. After writing, the variable which controls this, will be refreshed, or decrease in case of a writing cycle doesn't happened. For example if the program is executed every 1000 ms and the archiving process has to take place every 2000 ms, there will be one cycle where the archiving process is carried out and another where not, and this variable previously described is going to control this.

First of all, there is a condition to lock the execution of this function block, when the read of the whole archive is required (maybe from a USB stick, or maybe from the PC application), so there is a global variable in the structure Control which is activated and block this process.

This way, it is achieved that while all the records are being read, no more records are written.

Function Block *CreateArray()*

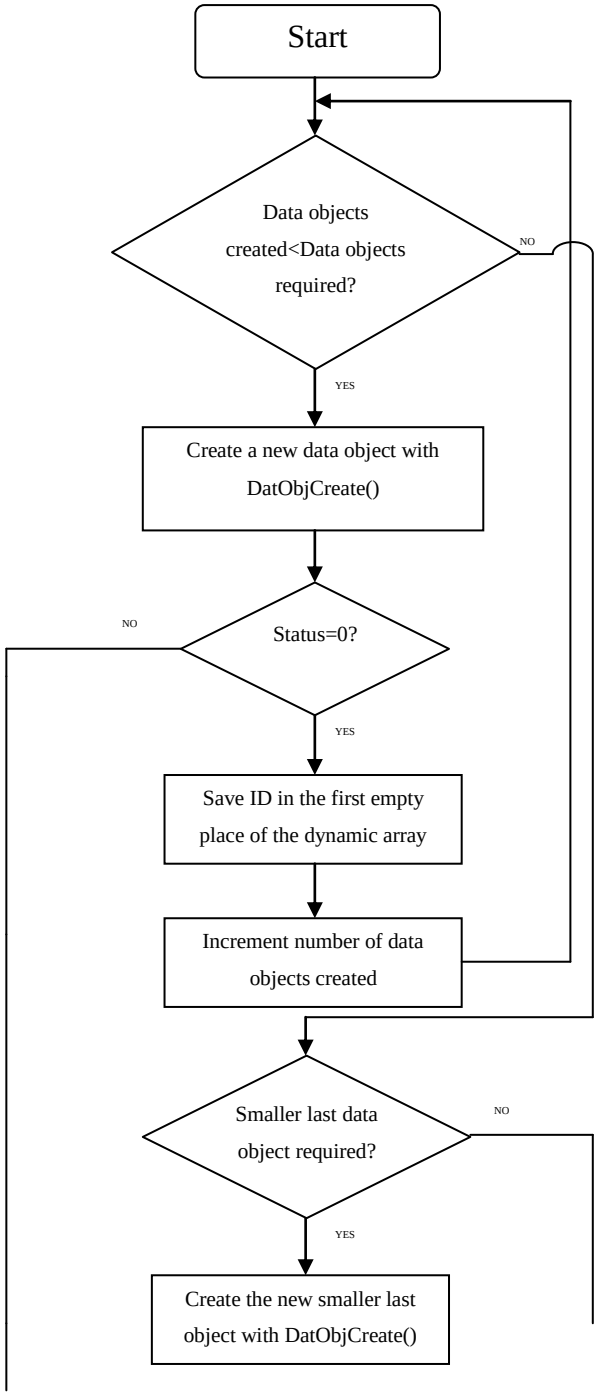
The mission of this block is to keep a number of free bytes in memory (System Ram), so for example, if we want an array to store ten addresses of 10 real values, taking into account that an address takes up into memory 4 bytes, we will have to pass to the input the value of 40 bytes to be reserved. This will be done by the programmer specifying what kind of data wants to save in the arrays and how many elements wants to archive.

It has two inputs (there will not be required to create more dynamic arrays at the same time) for the size of the arrays, and two outputs where the address of the first byte of the array will be placed. The number of free bytes necessary to create the dynamic arrays will be calculated with the parameters specified by the programmer, for example, if he want

It is not included the flowchart because of the simplicity. It is only necessary to pass the lenght of the arrays as parameters and then the function *TMP_Alloc()* will be used to allocate the memory. Finally, the address of the first element is sent out to the output variables.

Function Block *DObjCreation()*

This function block has two inputs, one variable type Control, and another with the size of the data objects that it has to create.



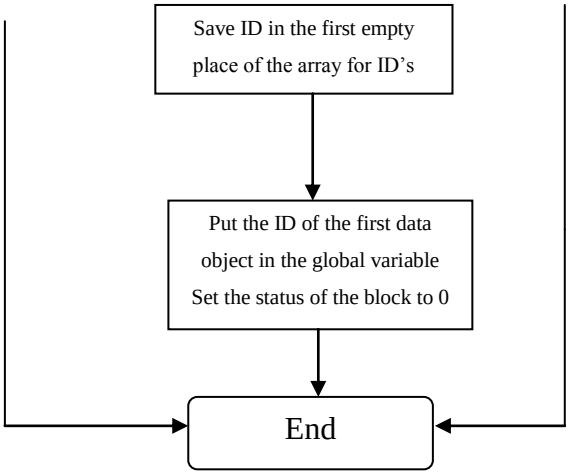
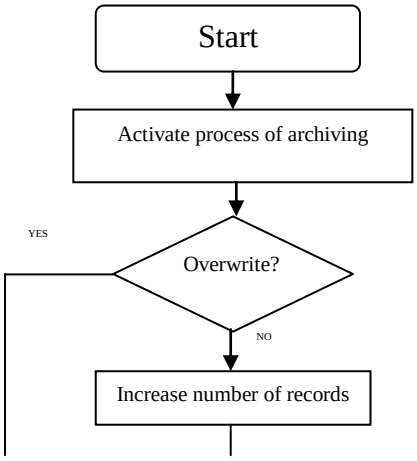


Figure 9. Flowchart of DobjCreation()

It consists of basically in one loop type FOR, this one is used for the creation of the data objects, so when one data object is created correctly, its ID is saved in the dynamic array created for the ID's of the objects. When the number of required objects have been created, it is checked if an additional smaller object is required, if so, it is created with a lenght previously calculated. Finally, the variable ID of the global variable type Control is set to the value to point the first data object, and the output variable status is set to 0 so the flow of the program can continue.

Function Block *DobjWriting()*

It has only one input type Control. The flow chart is shown below,



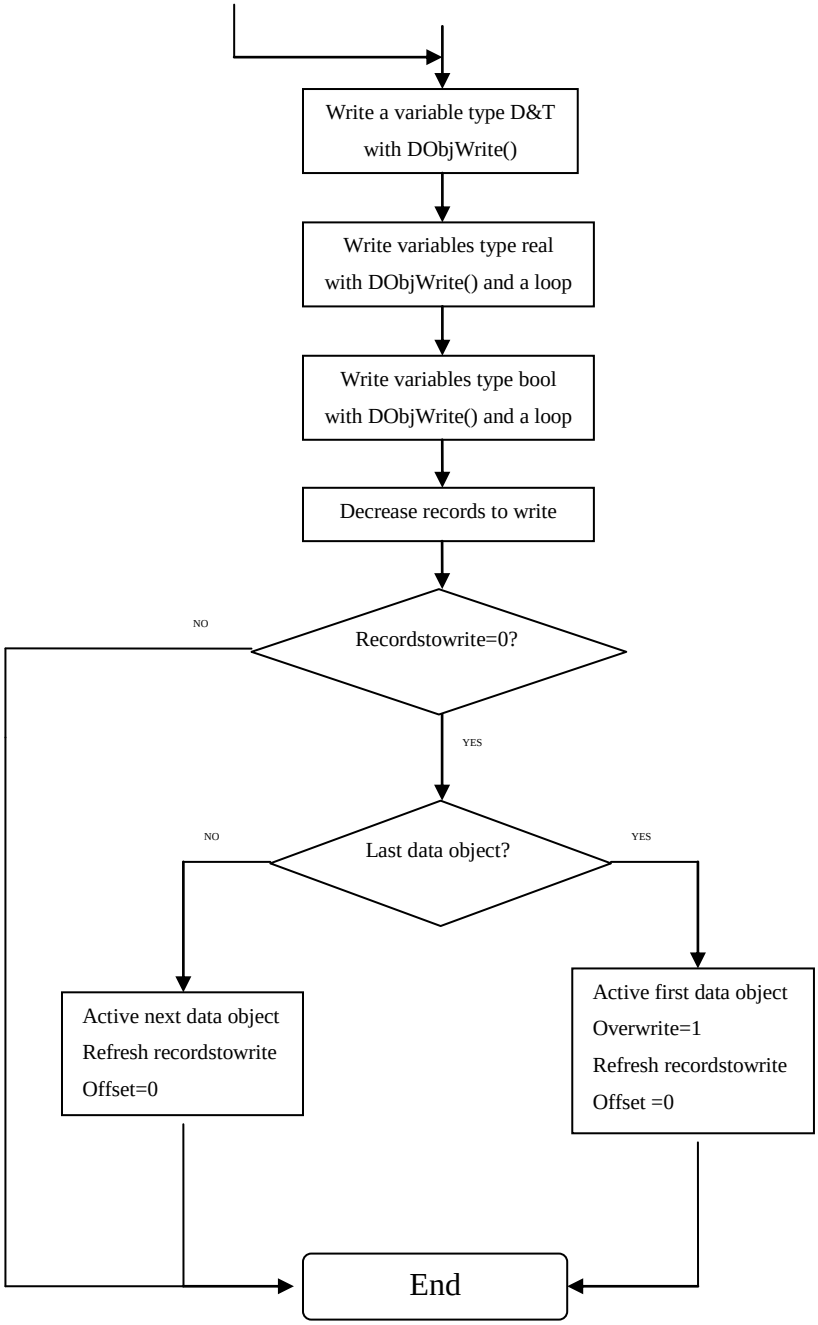


Figure 10. Flowchart of DobjCreation()

A little explanation of the processes of writing the real and boolean values. It consists of a *FOR* loop, where after each cycle, the offset is refreshed, so the values are written concatenated in the same data object. Due to the fact that the function block DatObjWrite() is not executed asynchronously (unlike MEMxInfo() or DatObjCreate()), it is possible to write the complete record in one cycle, repeating

the times which are needed the execution of *DatObjWrite()*. Each cycle, the address of the input data that has to be written changes to the address of the next input. The condition to finish is that both the real and the boolean values have been written.

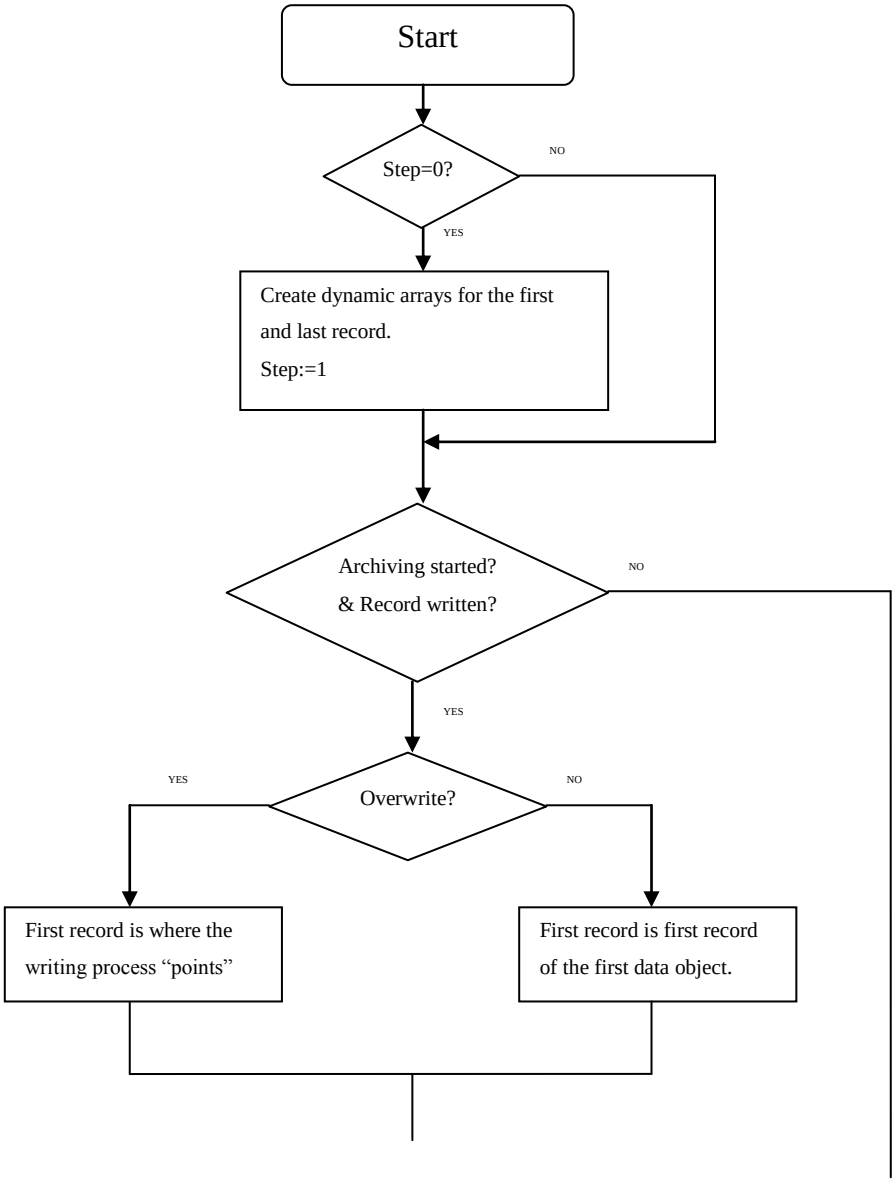
In the function block *Engine()* is calculated the exact number of data records that fit in the data object, so this parameter is decreased each time a record is written, so when this variable reachs the value of 0, it is checked if the active data object was the last or not. In case it was the last, the overwritting process is started. If there is a last smaller data object, this condition is also checked.

The number of records are considering the number included in the data objects, not the number since the system started working.

Function Block *ArchInfo()*

This function block is used to obtain some parameters which could be useful. It has as input, a variable type Control, and as outputs, a pointer to the last record, a pointer to the first record, the number of records in the archive, the size of one record, the period after which the archiving process takes place, and two pointers to the arrays where the addresses of the names are saved, one for the boolean and one for the real.

The flowchart is the following:



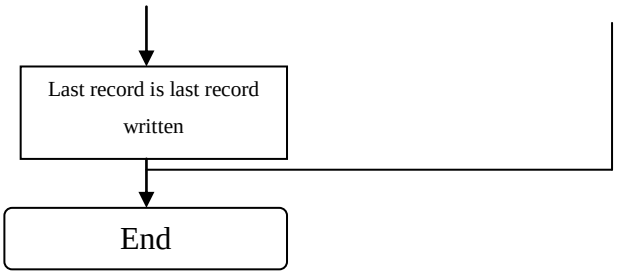


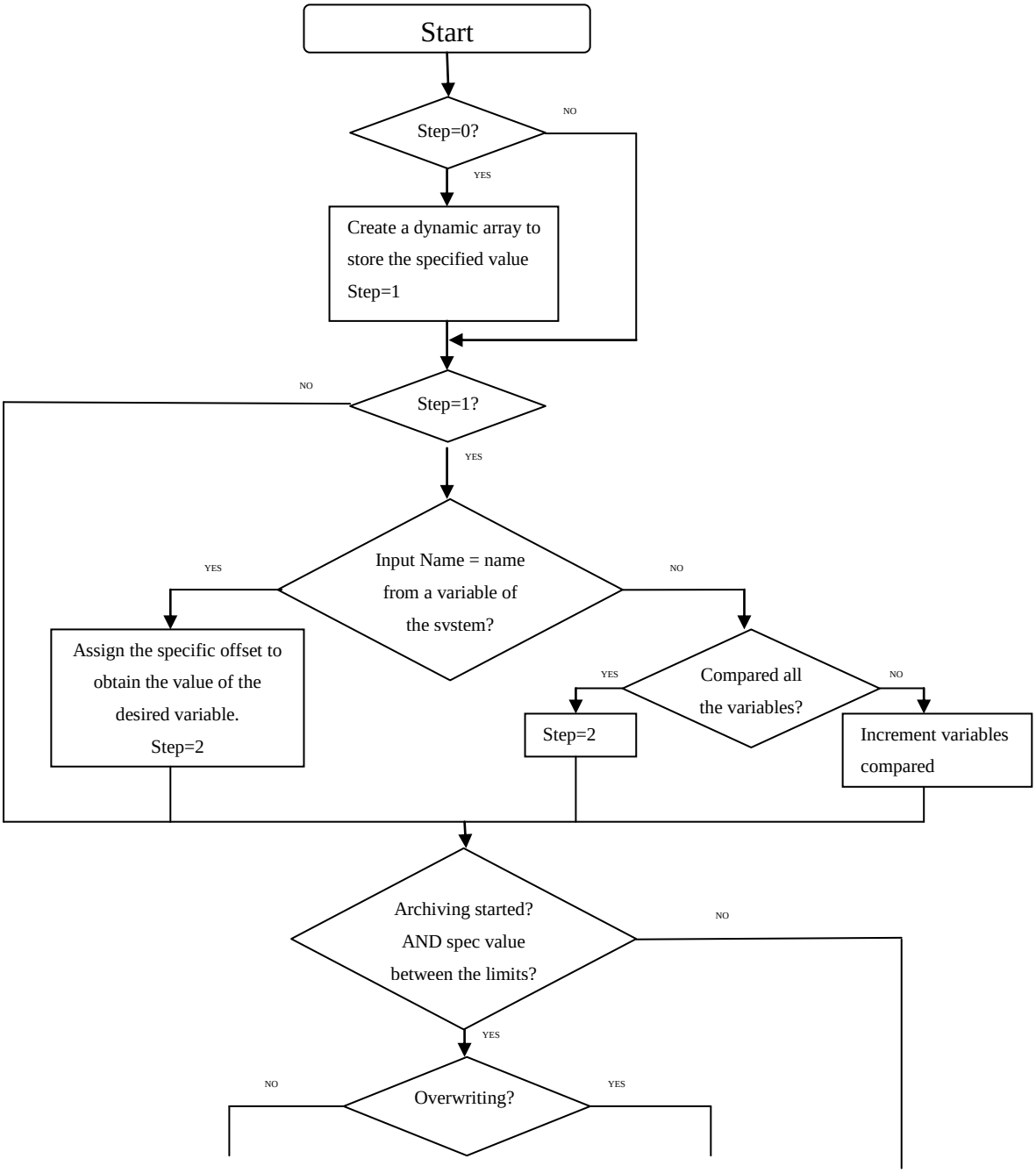
Figure 11. Flowchart of ArchInfo()

This function starts working when the archiving process starts, maybe after some cycles due to the fact that there are some function blocks which work asynchronously.

Note that when the arrays for the names are created, it is considered a maximum of 30 characters per variable connected.

Function Block *RecRead()*

This function is used to read a record specified for the programmer, so for example, if the programmer specifies the value 30, it is the 30th value starting from the oldest value. The inputs are a variable type Control and the desired value to read. There is another input which is a string to control a specific variable. The output is a pointer to the record and a pointer to the specific variable.



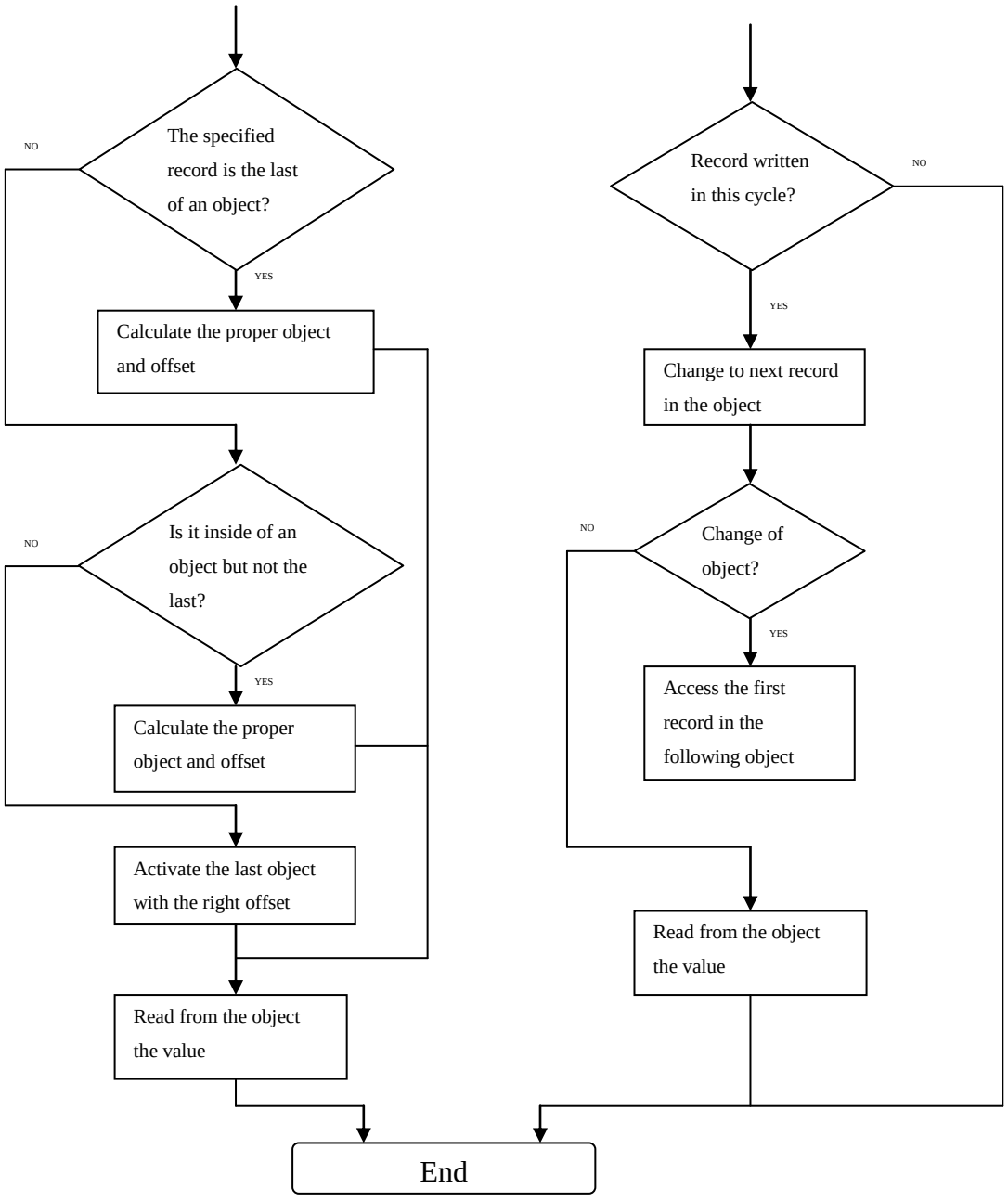


Figure 12. Flowchart of RecRead()

When it is said the if the value is valid, it means that it is checked if it is in the limits of the archiving, for example, if it is required the value number 200, and the archive is only ready to save 100 values, that would be an incorrect value.

Function Block *Online_Monitor()*

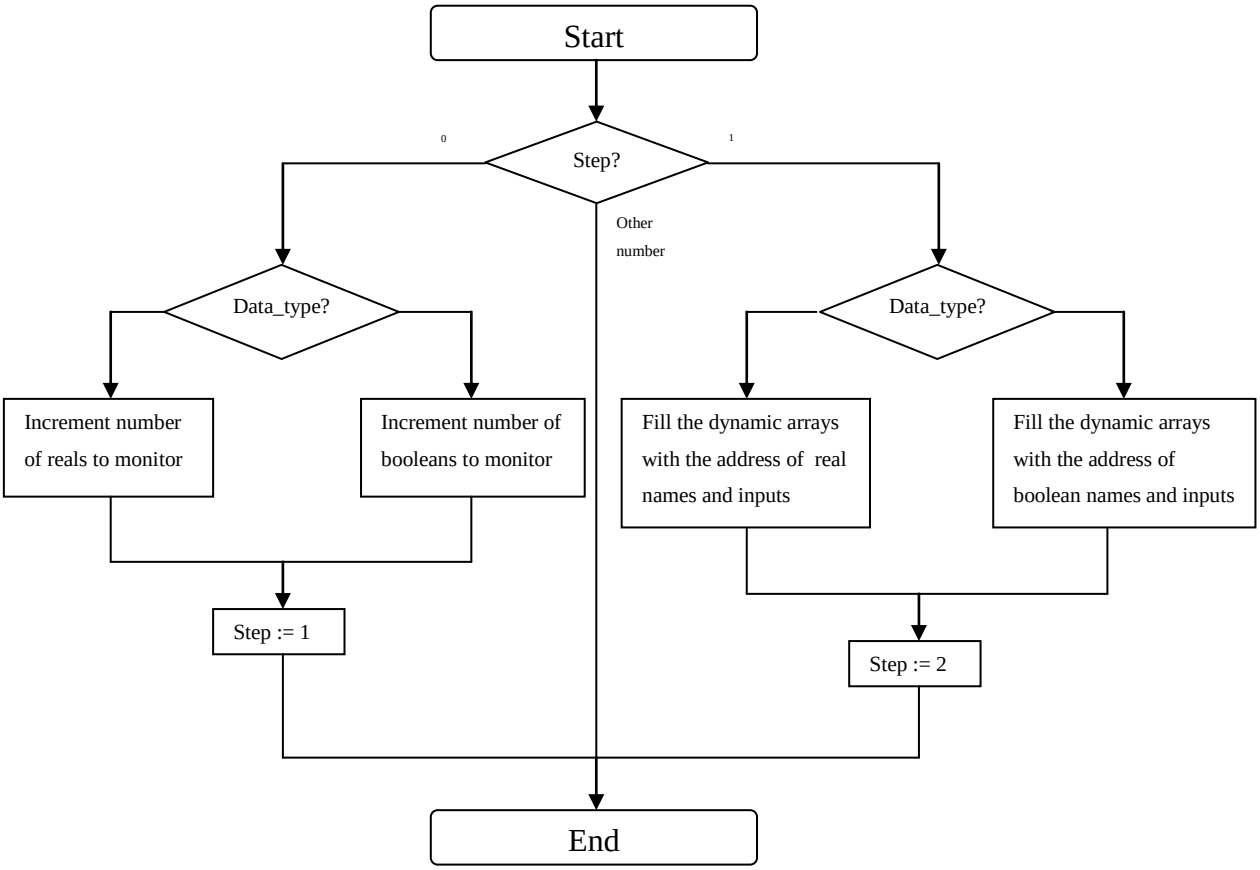
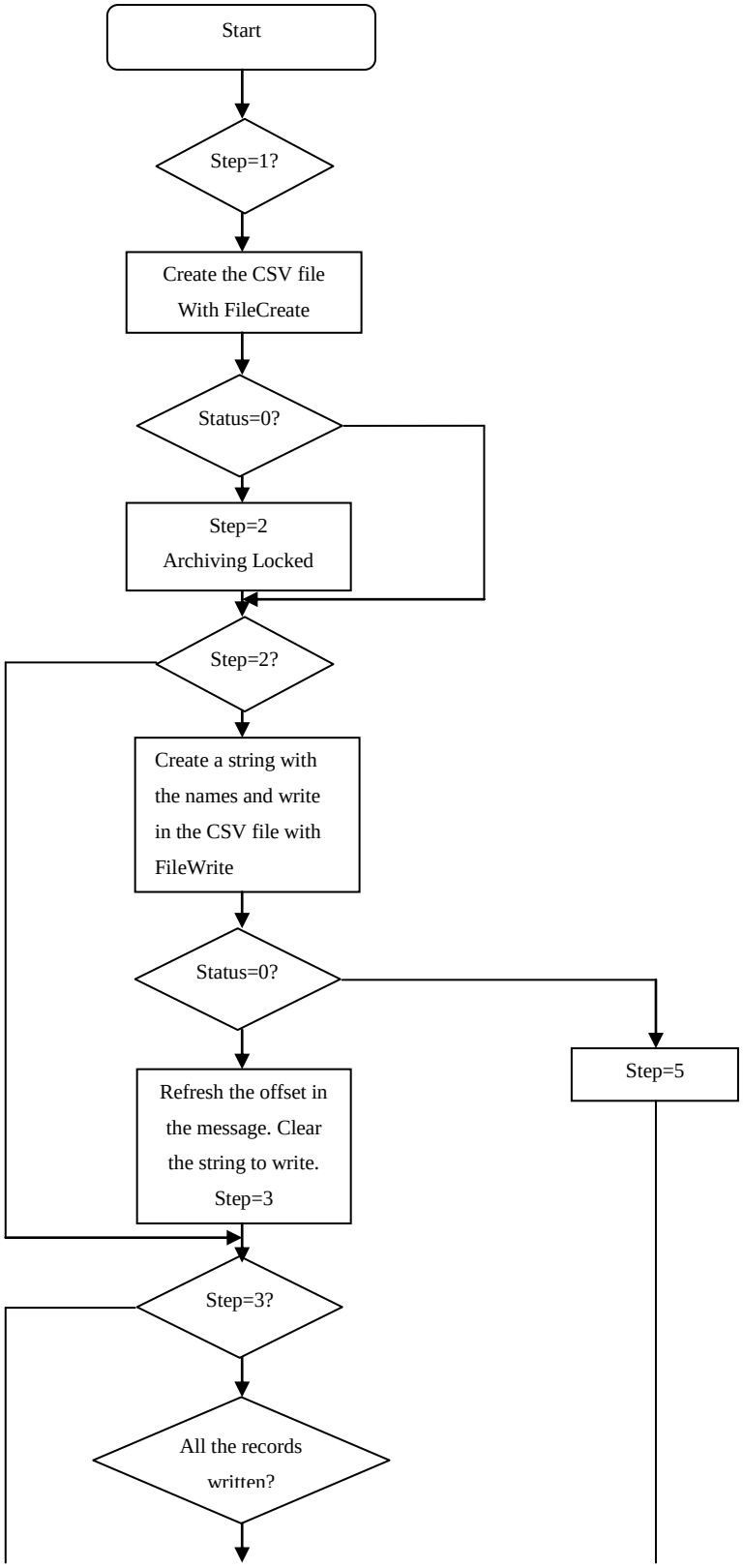


Figure 13. Flowchart of *Online_Monitor()*

This function block has 4 inputs, one for the name of the variable that is going to be monitored, another for the address of the input, boolean or real, one boolean to know if the connected variable is boolean or real, and finally one last type Control

Note that, all the variables that will be monitored later in the PC application must be connected to this function block.

Function Block *ArchivetoCSV()*



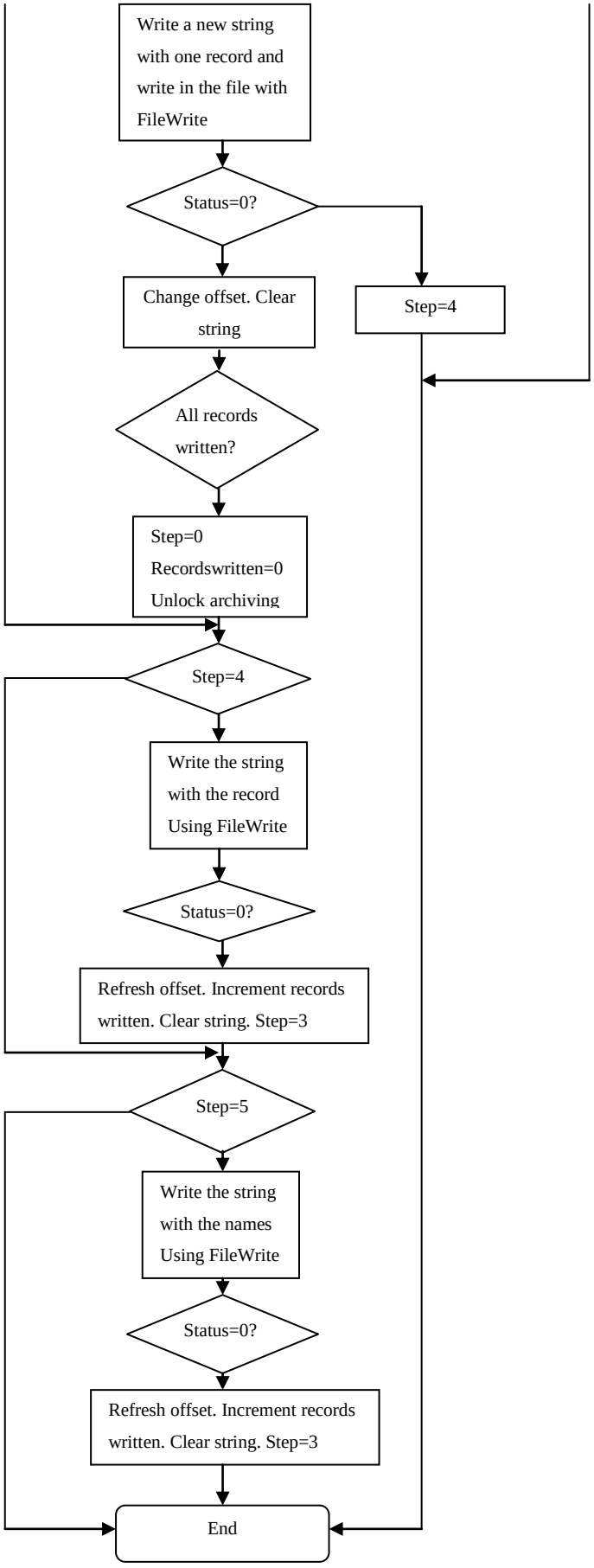
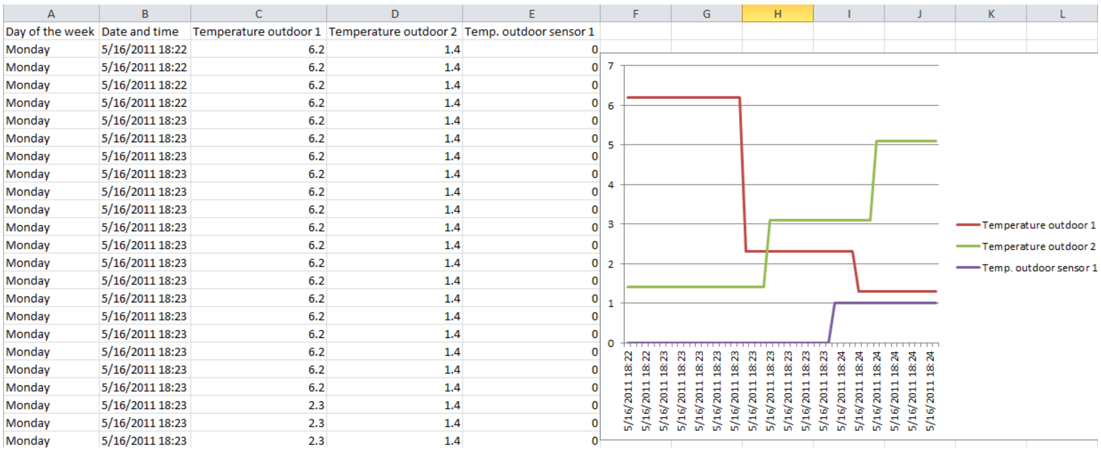


Figure 14. Flowchart of *ArchivetoCSV()*

As it was told earlier, this function block is used to create a CSV now in a memory card in the inserted in the PLC. This function block has only 1 input, this input is type Control. It should not be difficult modify so it could save the data in a USB inserted.

Note that the CSV is written in such a manner that it can be easily represented the data in a graph, it means that the date and time values, the reals and the booleans are written in the proper format. Here it is an example:



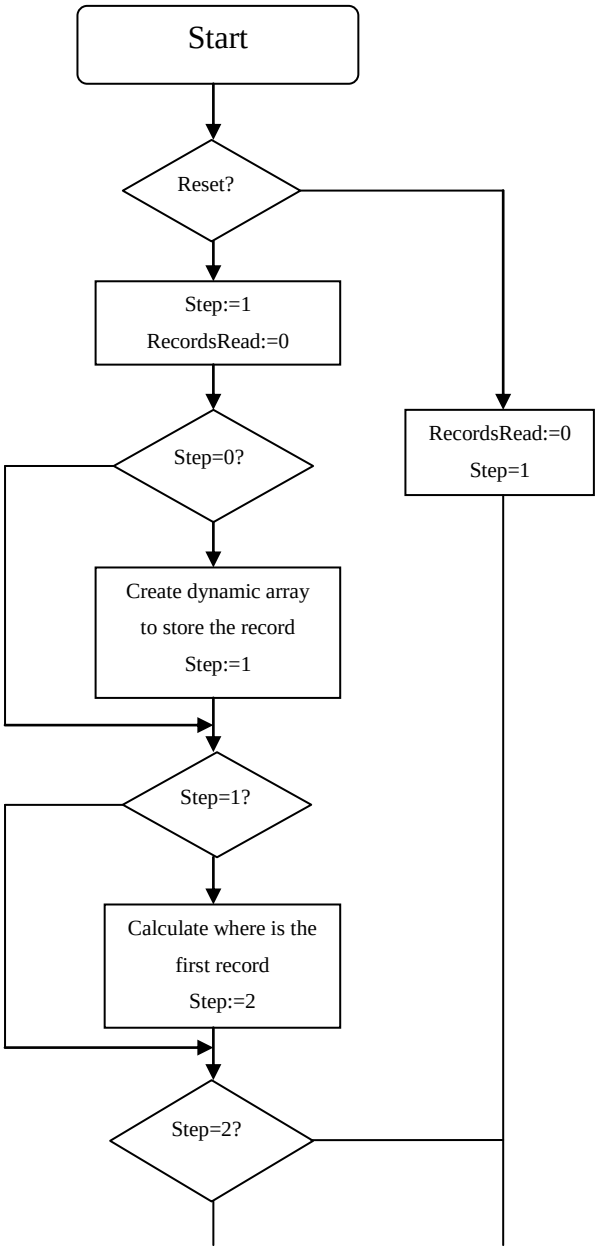
Day of the week	Date and time	Temperature outdoor 1	Temperature outdoor 2	Temp. outdoor sensor 1
Monday	5/16/2011 18:22	6.2	1.4	0
Monday	5/16/2011 18:22	6.2	1.4	0
Monday	5/16/2011 18:22	6.2	1.4	0
Monday	5/16/2011 18:22	6.2	1.4	0
Monday	5/16/2011 18:23	6.2	1.4	0
Monday	5/16/2011 18:23	6.2	1.4	0
Monday	5/16/2011 18:23	6.2	1.4	0
Monday	5/16/2011 18:23	6.2	1.4	0
Monday	5/16/2011 18:23	6.2	1.4	0
Monday	5/16/2011 18:23	6.2	1.4	0
Monday	5/16/2011 18:23	6.2	1.4	0
Monday	5/16/2011 18:23	6.2	1.4	0
Monday	5/16/2011 18:23	6.2	1.4	0
Monday	5/16/2011 18:23	6.2	1.4	0
Monday	5/16/2011 18:23	6.2	1.4	0
Monday	5/16/2011 18:23	6.2	1.4	0
Monday	5/16/2011 18:23	6.2	1.4	0
Monday	5/16/2011 18:23	6.2	1.4	0
Monday	5/16/2011 18:23	6.2	1.4	0
Monday	5/16/2011 18:23	6.2	1.4	0
Monday	5/16/2011 18:23	2.3	1.4	0
Monday	5/16/2011 18:23	2.3	1.4	0
Monday	5/16/2011 18:23	2.3	1.4	0

Figure 15. CSV opened with Excel

There are one more function block, this is and *ServerCreate()*, due to the fact that this two block is used to communicate the PC with the PLC, it will be described in a later section, see paragraph PC-PLC system.

Function Block *ReadArchive()*

This is a function block specific to read one record everytime it is executed, starting from the oldest one, to the last one. It has been created starting from the function block *RecRead()*. It has two inputs, one type *Control* and one bool called Reset, this input is used to „reset“ the function block in case it has not been read the whole archive, so next time after the reset that it will be executed, it will start again to read the oldest record in the archive.



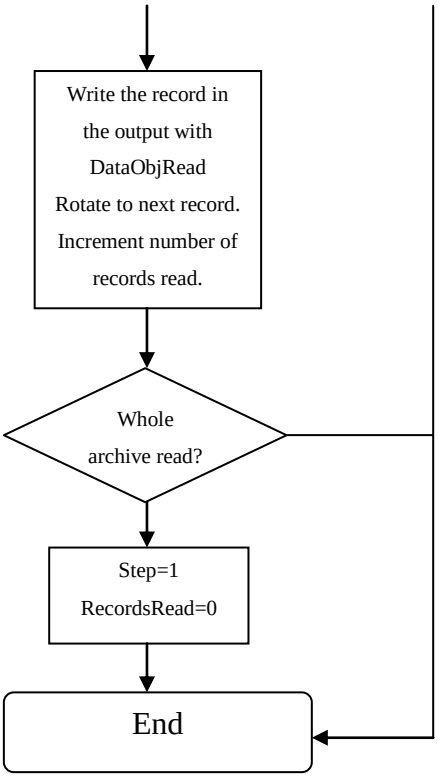


Figure 16. ReadArchive() flowchart

This block will be used to read the whole archive, both when a CSV file is required and when the whole archive is required by the PC application.

7.-TCP and UDP[4]

Due to the fact that we need a communication between the PLC and the PC, It is convenient to talk about the protocols used to manage the ports.

There are two types of Internet Protocol (IP) traffic, TCP and UDP. TCP (Transmission Control Protocol) is connection oriented, once a connection is established, data can be sent bidirectional. UDP (User Datagram Protocol) is a simpler, connectionless Internet protocol.

Differences in Data Transfer Features

TCP ensures a reliable and ordered delivery of a stream of bytes from user to server or vice versa. UDP is not dedicated to end to end connections and communication does not check readiness of receiver.

Reliability

TCP is more reliable since it manages message acknowledgment and retransmissions in case of lost parts. Thus there is absolutely no missing data. UDP does not ensure that communication has reached receiver since concepts of acknowledgment, time out and retransmission are not present.

Ordering

TCP transmissions are sent in a sequence and they are received in the same sequence. In the event of data segments arriving in wrong order, TCP reorders and delivers to application. In the case of UDP, sent message sequence may not be maintained when it reaches receiving application.

Connection

TCP is a heavy weight connection requiring three packets for a socket connection and handles congestion control and reliability. UDP is a lightweight

[4] The information here but the conclusión is extracted from wikipedia.[1][2]

transport layer designed atop an IP. There are no tracking connections or ordering of messages.

Differences in how TCP and UDP work

A TCP connection consists of three steps. The first is the process of initiating and acknowledging the connection. Once the connection is established, data transfer can begin. After transmission, the connection is terminated by closing of all established virtual circuits.

UDP uses a simple transmission model without guaranteeing realiability, ordering, or data integrity. UDP assumes that error checking and correction is either not necessary or performed in the application, avoiding the overhead of such processing at the network interface level. Unlike TCP, UDP is compatible with packet broadcasts (sending to all on local network) and multicasting (send to all subscribers).

Different applications of TCP and UDP

Web browsing, email and file transfer are typical TCP applicacions. TCP is used to control segment size, rate of data exchange, flow network congestion. TCP is preferred where error correction facilities are required at network interface level. UDP is largely used by time sensitive applications as well as by server that answer small queries from huge number of clients. UDP is compatible with packet broadcast. UDP is commonly used in Voice over IP, Trivial File Transfer Protocol and online games.

After this comparison, taking into account that our system can be considered like a non-time critical application and to ensure that all the data transferred remains intact and the data is received in the same order in which it was sent, it would be appropiate to use TCP.

8.- ETHERNET/IP and MODBUS/TCP

In the selection of an appropiate internet protocol to communicate the devices, these two are going to be studied. Only these two due to the fact that after checking the available possibilites in Internet to achieve the requierements, these two are the most important protocols nowadays.

Ethernet/IP and CIP[5]

Ethernet/IP is the application layer protocol. Four independent groups have joined forces to develop and promote EIP (Ethernet/IP) as a public domain Ethernet application layer for Industrial Automation. These groups include the ODVA, the Industrial Open Ethernet Association (IOANA), Control Net International (CI) and the Industrial Ethernet Association (IEA). The goals of this effort illustrate how EIP provides a wide-ranging, comprehensive, certifiable standard suitable to a wide variety of automation devices:

- Ethernet/IP uses the tools and technologies of traditional Ethernet. Ethernet/IP uses all the transport and control protocols used in traditional Ethernet including the Transport Control Protocol (TCP), the Internet Protocol (IP) and the media access and signaling technologies found in off-the-shelf Ethernet interface cards. Building on these standard PC technologies means that EIP works transparently with all the standard off-the-shelf Ethernet devices found in today's marketplace. It also means that EIP can be easily supported on standard PCs and all their derivatives. Even more importantly, basing EIP on a standard technology platform ensures that EIP will move forward as the base technologies evolve in the future.

[5] All this information is from wiki (EtherNet/IP) and from the website of the company Real Time Automation.[7][10]

- Ethernet/IP is a certifiable standard. The groups supporting EIP plan to ensure a comprehensive, consistent standard by careful, multi-vendor attention to the specification and through certified test labs as has been done with DeviceNet and ControlNet. Certification programs modeled after the programs for DeviceNet and ControlNet will ensure the consistency and quality of field devices.

- EIP is built on a widely accepted protocol layer. EIP is constructed from a very widely implemented standard used in DeviceNet and ControlNet called the Common Industrial Protocol (CIP). This standard organizes networked devices as a collection of objects. It defines the access, object behavior and extensions which allow widely disparate devices to be accessed using a common mechanism. Hundreds of vendors now support the CIP protocol in present day products. Using this technology in EIP means that EIP is based on a widely understood, widely implemented standard that does not require a new technology shakedown period.

The Common Industrial Protocol (CIP) is a communications protocol for transferring automation data between two devices. In the CIP Protocol, every network device represents itself as a series of objects. Each object is simply a grouping of the related data values in a device. For example, every CIP device is required to make an Identity object available to the network. The identity object contains related identity data values called attributes. Attributes for the identity object include the vendor ID, date of manufacture, device serial number and other identity data. CIP does not specify at all how this object data is implemented, only what data values or attributes must be supported and that these attributes must be available to other CIP devices.

The Identity object is an example of a required object. There are three types of objects defined by the CIP protocol:

Required objects are required by the specification to be included in every CIP device. These objects include the Identity object, a Message Router object and a Network object.

A. The identity object contains related identity data values called attributes. Attributes for the identity object include the vendor ID, date of manufacturer, device serial number and other identity data.

B. The Message Router object is an object which routes explicit request messages from object to object in a device.

C. A Network object contains the physical connection data for the object. For a CIP device on DeviceNet the network object contains the MacID and other data describing theinterface to the CAN network. For EIP devices, the network object contains the IP address and other data describing the interface to the Ethernet port on the device.

Application objects are the objects that define the data encapsulated by the device. These objects are specific to the device type and function. For example, a Motor object on a Drive System has attributes describing the frequency, current rating and motor size. An Analog Input object on an I/O device has attributes that define the type, resolution and current value for the analog input.

These application layer objects are predefined for a large number of common device types. All CIP devices with the same device type (Drive Systems, Motion Control, Valve Transducer…etc) must contain the identical series of application objects. The series of application objects for a particular device type is known as the device profile. A large number of profiles for many device types have been defined. Supporting a device profile allows a user to easily understand and switch from a vendor of one device type to another vendor with that same device type.

A device vendor can also group Application Layer Objects into assembly objects. These super objects contain attributes of one or more Application Layer

Objects. Assembly objects form a convenient package for transporting data between devices. For example, a vendor of a Temperature Controller with multiple temperature loops may define assemblies for each of the temperature loops and an assembly with data from both temperature loops. The user can than pick the assembly that is most suited for the application and how often to access each assembly. For example, one temperature assembly may be configured to report every time it changes state while the second may be configured to report every one-second regardless of a change in state.

Assemblies are usually predefined by the vendor but CIP also defines a mechanism in which the user can dynamically create an assembly from application layer object attributes.

Vendor specific objects not found in the profile for a device class are termed Vendor Specific. These objects are included by the vendor as additional features of the device. The CIP protocol provides access to these vendor extension objects in exactly the same method as either application or required objects. This data is strictly of the vendors choosing and is organized in whatever method makes sense to the device vendor. In addition to specifying how device data is represented to the network, the CIP protocol specifies a number of different ways in which that data can be accessed such as cyclic, polled and change-of-state.

Modbus TCP[6]

ModbusTCP is an open protocol derived from the Master/Slave architecture. It runs on a Ethernet physical layer. It is a widely accepted protocol due to its ease of use and reliability. This wide acceptance is due in large part to MODBUS TCP‘s ease of use. Modbus RTU is widely in Industrial Automation Sytems (IAS).

[6] Except the conclusion, this information has been obtained from the specification of Modbus/TCP and from some little documents in the web of the companies Real Time Automation and Intellicom.[3][4][8]

MODBUS is considered an application layer messaging protocol, providing Mastert/Slave communication between devices connected together through buses or networks. On the OSI model, MODBUS is positioned at level 7. MODBUS is intended to be a request/reply protocol and delivers services specified by function codes. The function codes of MODBUS are elements of MODBUS' request/reply PDUs (Protocol Data Unit).

In order to build the MODBUS application data unit, the client must initiate a MODBUS transaction. It is the function which informs the server as to which type of action to perform. The format of a request initiated by a Master is established by the MODBUS application protocol. The function code field is then coded into one byte. Only codes within the range of 1 through 255 are considered valid, with 128-255 being reserved for exception responses. When the Master sends a message to the Slave, it is the function code field which informs the server of what type of action to perform.

To define multiple actions, some functions will have sub-function codes added to them. For instance, the Master is able to read the ON/OFF states of a group of discreet outputs or inputs. It could also read/write the data contents of a group of MODBUS registers. When the Master receives the Slave response, the function code field is used by the Slave to indicate either an error-free response or an exception response. The Slave echoes to the request of the initial function code in the case of a normal response.

MODBUS TCP messages are not a 16-bit CRC (Cyclic-Redundant Checksum) like in MODBUS RTU, the TCP/IP and link layer (eg. Ethernet) checksum mechanisms instead are used to verify accurate delivery of the packet.
Modbus/TCP basically embeds a Modbus frame into a TCP frame in a simple manner. This is a connection-oriented transaction which means every query expects a response.

This query/response technique fits well with the master/slave nature of Modbus, adding to the deterministic advantage that Switched Ethernet offers industrial users. The use of OPEN Modbus within the TCP frame provides a totally scaleable solution from ten nodes to ten thousand nodes without the risk of compromise that other multicast techniques would give.

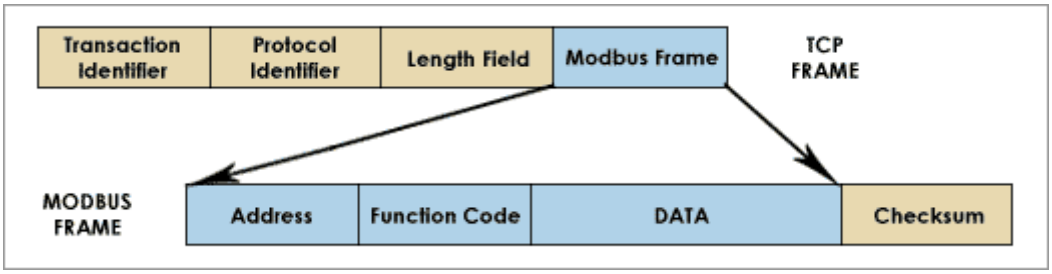


Figure 17. Structure of how a Modbus frame fits in a TCP frame.

The performance basically depends on the network and the hardware. If it is running MODBUS® TCP/IP over the Internet, it won't get better than typical Internet response times. However, for communicating for debug and maintenance purposes, this may be perfectly enough.

Finally, before a conclusion, the requirements for the protocol will be that it could send or receive all data types in the easiest way possible, to obtain this, they could be transformed before and after being sending.

Due to the fact that the two protocols obviously will achieve this requirement, to compare the two protocols, the analysis should focus on the price and simplicity.

Modbus is an open protocol, so it is completely free to use and to be implement, whereas Ethernet/IP is not an open protocol, so it is not free. The cost due to the implementation are also higher.

After reading the description of the Ethernet/IP protocol, it seems to be a little complex protocol, because although it has a lot of possibilities, it also has a lot of unnecessary things for our application (several kinds of objects, it is possible to send resolution, units….). Whereas Modbus, its only and main application is to send data which is our mission.

It can be concluded that to achieve the objectives it will be used Modbus or any similar protocol that it could be created from Modbus.

An example of how can be sent a real value or a boolean value with Modbus is shown below:

The most efficient method of transporting bulk information or any type over MODBUS is as to use function codes 3 (read registers), 16 (write registers) or possibly 23 (read/write registers).

Although these functions are defined in terms of their operation on 16-bit registers, they can be used to move any type of information from one machine to another, so long as that information can be represented as a contiguous blocks of 16-bit words.

Almost all data types other than the primitive “discrete bit” and “16 bit register” were introduced after the adoption of little-endian microprocessors. Therefore the representation on MODBUS of these data types follows the little-endian model, meaning

First register bits 15 – 0 = bits 15 – 0 of data item
Second register bits 15 – 0 = bits 31 – 16 of data item
Third register bits 15 – 0 = bits 47 – 32 of data item
Etc, etc

For example, to send a Boolean value is a 1-bit quantity, so the bit 0 of the register will be the bit of the Boolean value.

On the contrary, if it is a Real value (32-bit quantity), it has to be used two registers, bits 15 to 0 of the first register will be for bits 15 to 0 of the Real value, and bits 15 to 0 of the second register will be for bits 31 to 16 of the Real value (exponent + bits 23-16 of significand).

9.- PLC – PC system

General description

The process controlled by the system has to be monitored in real time, so the communication between the devices must be continuous.

Thus, a HMI (human interface) will be created to let the user control the system, this interface will be really friendly because it is assumed that the user has no knowledge about the system.

As it was concluded in the previous sections, the communication between the PLC and the PC is going to be carried out by means of a TCP protocol, this protocol will be developed from the Modbus specification. First of all, some basic concepts about communication between devices are shown.

The model implemented is a typical Client/Server model, where the client is the PC and the server is the PLC. So the client can send requests everytime it wants, and the server will send the reponses.

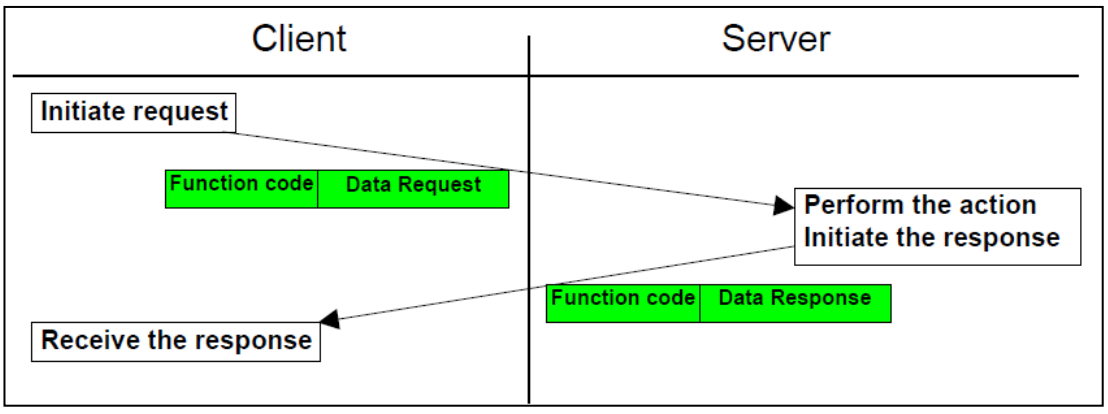


Figure 18. Typical request - reponse.

The request is the message sent on the network by the client to initiate a transaction. The reponse is the message sent by the server.

The steps followed to achieved the communication between the devices are shown below:

Client (C# Windows forms)

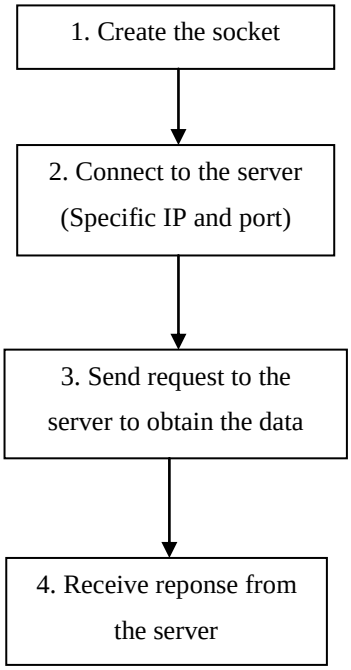


Figure 19. Client operation.

So, these are the main steps in the client to get some information from the server. The two last steps are repeated while there are more information required.

Before a socket can be used to communicate with remote devices, the socket must be initialized with protocol and network address information. The constructor for the *Socket class* has parameters that specify the address family, socket type, and

protocol type that the socket uses to make connections. To create our sockets, due to the fact that we communicate on a TCP/IP-based network, we set the parameters *AddressFamily* (standard address families used by the *Socket class* to resolve network addresses) to InterNetwork to specify the IP version 4 address family. The *SocketType* specifies the type of the socket, it will be selected stream, to indicate a standard socket for sending and receiving data with flow control. Finally, the last parameter to create the socket is *ProtocolType* that specifies the network protocol to use when communicating on the socket, as we use a TCP protocol, the option TCP will be selected. An example is as follows:

Socket s = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

When the socket is created, it has to be connected to a remote host (PLC). This is specified with a IP address and a port. To achieve this, it is used the method *Connect()* from the *Socket* class. If the connection is not possible an exception is generated (More information about exceptions in MSDN site).

Now it is time send the specified request to the server, the proper method is *Send()*. With this method a stream of bytes is sent to the server.

Finally, the reponse from the server is received with the method *Receive()*. This way, the data from the server is received in a stream of bytes which works as a buffer.

A detailed description of the PC application in C# Windows Forms is given later (see section PC application).

Server (PLC application - ST)

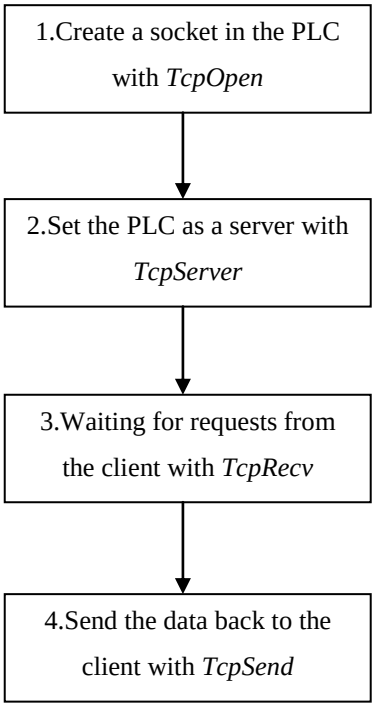


Figure 20. Server operation.

When the server is configured, two first steps, and after processing a request and send back the data to the client, it returns to the step 3.

First of all, a TCP socket is created with the function block *TcpOpen().* It needs some parameters, the IP address of the Ethernet interface where the TCP socket should be connected. This is going to be configured in such a manner that it will be listening in all the interfaces. Another parameter is the port number. There is another possible option to set that allows the binding of several instances of a TCP server with the same port, but it will no be activated.

After the TCP socket is created, it is configured as a server. This is done by means of the function block *TcpServer().* The parameters here are ident (taken when

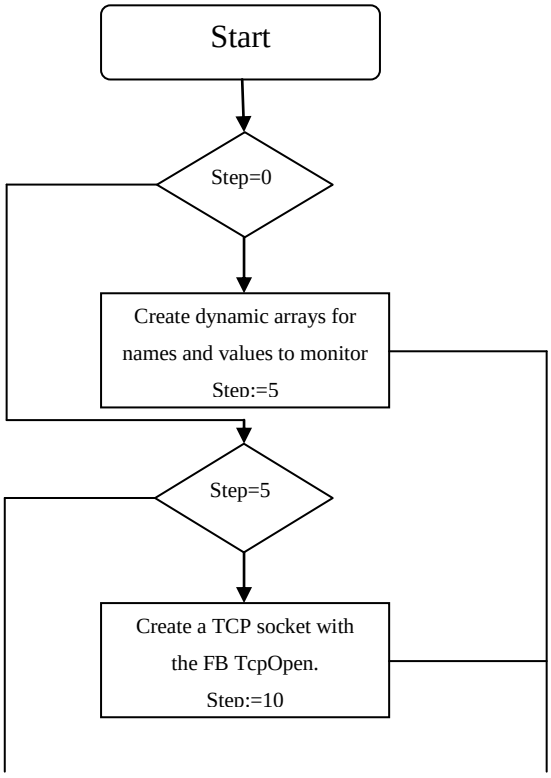
the socket is opened *TcpOpen()*), backlog, which are the number of clients waiting simultaneously for a connection, 1.

Now the system is ready to receive requests from the client, so it will be waiting for data (executing cyclicly the function block *TcpRecv*). It needs the client ident, a pointer to the data buffer and the max. Lenght of the data in bytes.

Finally, when the request is processed, it is sent back to the client with the function block *TcpSend*.

To achieve this, there is one function block, called ***ServerCreate***, in the library created for the remote management in the PLC. This function block has two inputs, one type Control, and another type UDINT, to specify the port where to create the socket for the server. It also has an output, this output is to check the right working of the system, and in case of one error, it is possible to know where that error is.

The flow chart of the function block is shown below:



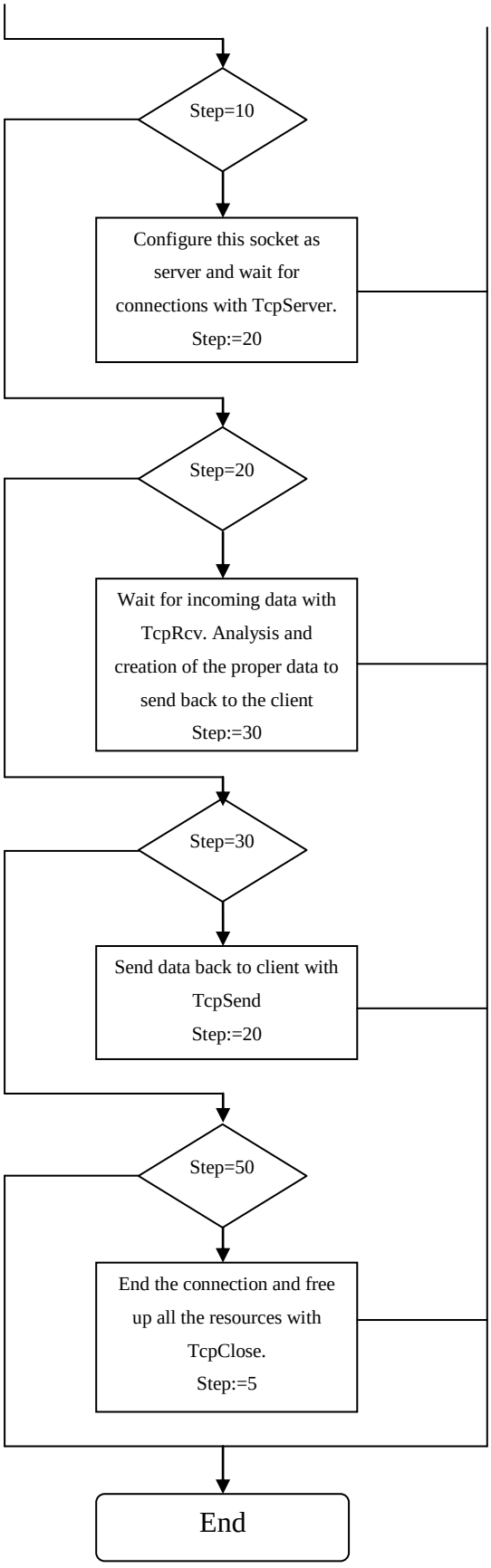


Figure 21. Flowchart of *ServerCreate()*

The execution of this function block is controlled with the variable Step. After the execution of every block, for example, when some data is received from the client, the status of the function block TcpRcv is analyzed to know if some data has to be sent back to the client, or some error has ocurred.

This way, this block has an special functionality which could be extrapolated to some other function blocks in the library. The status output of each function block used inside this one has been analyzed, and depending on these values, the status output of the whole block (*ServerCreate* in this case), is set to one different value.

This status of the whole function can be used to write in the logger, which is a register where all the events are stored, different messages according to the value of this status output. The correspondence between the values of the output status and the status of the function blocks inside *ServerCreate* is shown as follows:

Error (*ServerCreate-FB)*	FB where the error is	Description
1-32601	TcpOpen()	Could not reserve additional ident
2-32602		Socket already connected to this port number
3-32650		Resource problem in system. Could not create new sockets.
4-32651		Problem binding to the port number or IP address
5-32612		The specified interface address is invalid.
6-32601	TcpServer()	Could not reserve additional ident. A new ident is needed for each new TCP client.
7-32652		Error listening to the socket.
8-32653		Error accepting on the socket.
9-32600	TcpRecv()	The specified filename is not allowed.
10-32609		The connection is closed (opposite station).
11-32699		Internal error during sending.
12-32600	TcpSend()	The specified filename is not allowed.
13-32603		The *pData* data pointer is not set (0).
14-32606		The data length applied doesn't correspond to the specified data length.
15-32607		Data could not be applied in the socket's send buffer.
16-32609		The connection is closed (opposite station).
17-32699		Internal error during sending.
18-32600	TcpClose()	The specified filename is not allowed.

Table 3. Values and errors in *ServerCreate()*

An example of how to write a message in the logger with the function block *AsArLogWrite()* is shown below:

```
(*Write a log entry *)
Logger.AsArLogWrite_0.enable := 1;
Logger.AsArLogWrite_0.errornr := 32601;                (*User Error Number*)
Logger.AsArLogWrite_0.ident := 1;                      (*Ident of the AR logger user module*)
Logger.AsArLogWrite_0.logLevel := 2;                   (*Loglevel of the entry: 1 = Information*)
Logger.AsArLogWrite_0.mem := 0;                        (*additional binary data*)
Logger.AsArLogWrite_0.len := 0;                        (*Length of the binary log data in bytes*)
Logger.AsArLogWrite_0.asciiString := ADR(String);  (*Log-specific zero-terminated ASCII string*)
Logger.AsArLogWrite_0;                                 (* Call the Functionblock*)
```

Figure 22. Code to use *AsArLogWrite* function

The input parameter called errornr (Error number), the number obtained in the help from Automation Studio can be set, and then the description in the logger will be written. In this example, the error number is 32601, so in the logger will be written „Could not reserve.....“. The binary data is set 0, it is not considered that will be useful for the programmer this kind of information. The variable *String* will be used to give more information about the error, so the programmer will know exactly where the error is inside the block *ServerCreate*. For more information about this function block, the help from Automation Studio can be consulted.

10.- Communications frames used

A group of communication frames are needed in order to send the data stored in the PLC to the PC. The proposed frames and their descriptions are as follows:

Strings

Request

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte (Operation)	5º - 8º byte (ID)
1	1	5	0 or 1	

Fig 23. Frame to request string

Byte range	Name	Description
1	PLC – ID	This parameter is used to identify the PLC that we are working with.
2	Protocol Version	This system will work only with the right version of the protocol, 1 in this case.
3	Lenght	Here it is possible to know the lenght of the message, that is, the number of bytes after this 3º byte.
4	Operation	This byte is used to identify what operation to develop. In this case is used 0 or 1 to request real or boolean names, respectively.
5-8	ID	It is possible to identify which variable is required. Because they are four bytes, it is possible to identify up to 4294967296 variables.

Table 4. Request string description

Reponse

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte	5º - nº byte
1	1	Lenght of the string +1	70	Data

Fig 24.Reponse frame with a string

Byte range	Name	Description
3	Lenght	The lenght of the reponse will be given in this case by the lenght of the string and one more byte.

4	Reponse code	This will be a byte to know that the reponse has been created correctly. The value is 50 in this case.
5-n	Data	This will be variable, because the lenght of a string is variable.

Table 5. Reponse string description

Reals

Request

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte (Operation)	5º - 8º byte (ID)
1	1	5	2	

Fig 25. Frame to request a real

Byte range	Name	Description
4	Operation	In this case this byte gets the value 2. This is to identify that a real value is required.

Table 6. Request real description

Reponse

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte	5º - 8º byte
1	1	5	60	Data

Fig 26. Reponse frame with a real

Byte range	Name	Description
4	Reponse code	This will be a byte to know that the reponse has been created correctly. The value is 40 in this case.
5-8	Data	The data is fixed in this case to 4 bytes, because of the lenght of a real value, 4 bytes.

Table 7. Request real description

Booleans

Request

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte (Operation)	5º - 8º byte (ID)
1	1	5	3	

Fig 27. Frame to request a boolean value

Byte range	Name	Description
4	Operation	In this case this byte gets the value 2. This is to identify that a boolean value is required.

Table 8. Request bool description

Reponse

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte	5º byte
1	1	2	50	Data

Fig 28. Reponse frame with a boolean value

Byte range	Name	Description
4	Reponse code	This will be a byte to know that the reponse has been created correctly. The value is 30 in this case.
5-8	Data	The data is fixed in this case to 1 bytes, because of the lenght of a real value, 1 byte.

Table 9. Reponse boolean description

To request the list of variables to be monitored

Request

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte (Operation)
1	1	1	4

Fig 29. Frame to request the list of variables to be monitored

Byte range	Name	Description
4	Operation	In this case this byte gets the value 4. This is to require the list of variables available to be monitored.

Table 10. Request for variables to monitor description

In this case it is not necessary an ID.

Reponse

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte	5º-6º byte	7º-8º byte
1	1	5	40	Nº Real values	Nº Boolean values

Fig 30. Reponse frame with the variables to be monitored

Byte range	Name	Description
4	Reponse code	This will be a byte to know that the reponse has been created correctly. The value is 20 in this case.
5-6	Nº Real values	The number of real values available for online monitoring.
7-8	Nº Boolean values	The number of boolean values available for online monitoring.

Table 11. Reponse variables to monitor description

To request the whole archive

Request

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte (Operation)
1	1	1	5

Fig 31. Frame to request the whole archive

Byte range	Name	Description
4	Operation	In this case this byte gets the value 5. This is to require the whole archive.

Table 12. Request whole archive description

Reponse

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte	5º-6º byte	7º-8º byte	9º-10º byte
1	1	7	30	Nº of records	Nº of reals	Nº of booleans

Fig 32. Reponse frame with the values needed for the whole archive

Byte range	Name	Description
4	Reponse code	This will be a byte to know that the reponse has been created correctly. The value is 30 in this case.
5-6	Number of records	This is the number of records existing currently in the archive.
7-8	Number of reals	This is the number of real variables in the archive
9-10	Number of booleans	The number of boolean variables in the archive

Table 13. Reponse whole archive description

To request the records from the whole archive

Request

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte (Operation)
1	1	1	6

Fig 33. Frame to request one record from the archive.

Byte range	Name	Description
4	Operation	In this case this byte gets the value 6. This is to require one record from archive.

Table 14. Request one record description

Reponse

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte	5º byte	6º - n byte
1	1	Lenght of the record + 2	20	Lenght of the string with D&T	Record

Fig 34. Reponse frame with one record from the archive

Byte range	Name	Description
4	Reponse code	In this case this byte gets the value 20. This is to know that a record has been received.
5 - n	Record	The record required is received in the reponse.

Table 15. Reponse one record description

Here, it is important to know that the Date&Time variable in the reponse will be in a string.

To request the names of the variables of the whole archive

Request

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte (Operation)
1	1	1	7

Fig 35. Frame to request the name of a variable of the system

Byte range	Name	Description
4	Operation	In this case this byte gets the value 7. This is to require the name of a variable from the whole archive.

Table 16. Request one variable name description

Reponse

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte	5º -n byte
1	1	Lenght of the string + 1	10	Data

Fig 37. Reponse frame to request the name of a variable of the system

Byte range	Name	Description
4	Reponse code	In this case this byte gets the value 10. This is to know that a name has been received.
5 - n	String	The string with the name of a variable is received in the reponse.

Table 17. Request one variable name description

To check the connection is still working

Request

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte (Operation)
1	1	1	8

Fig 38. Frame to know if the connection is still alive

Byte range	Name	Description
4	Operation	In this case this byte gets the value 8. This is to check the connection is still alive

Table 18. Request check the connection description

Reponse

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte
1	1	1	0

Fig 39. Reponse frame to know if the connection is still alive

Byte range	Name	Description
5	Reponse code	In this case this byte gets the value 0. This is to check a reponse so the client knows the connection is still alive.

Table 19. Reponse check the connection description

To obtain the values of all the variables available to be online monitored

Request

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte (Operation)
1	1	1	9

Fig 40. Frame to get the values from the online variables

Byte range	Name	Description
4	Operation	In this case this byte gets the value 9. This is to request all the values availabe to be online monitored.

Table 18. Request obtain the online values

Reponse

1º byte (ID PLC)	2º byte (Version)	3º byte (Lenght)	4º byte	5º byte	6º - n byte
1	1	Lengh of the values + 2	80	Lenght of the D&T string	Values to OM

Fig 41. Reponse with the values of the online variables

Byte range	Name	Description
4	Reponse code	In this case this byte gets the value 80. This is to check the data has been properly received.
5	Lenght of D&T string	This is important because this parameter is variable, it will be important to know where the values of the variable start.

Table 19. Reponse to the request of all the online values

First of all, a complete process of how to request the data from the whole archive is explained. When this mode is selected, a request like the one specified before (fig 31) is sent. The server receives this request, and then create a reponse with the number of variables, both real and boolean (fig 32), and the number of records currently in the archive, the archiving process in the PLC is locked until this transfering process of the whole archive has finished. It is sent to the client, and now the client with these values, fix the size of 4 arrays, 2 bidimensional for the values of the real and boolean values, one for the names and another one for date and time.

Now, the client sends a group of requests (fig 35) to get the names of the variables from the server (with a for loop). This loop is limited by the total number of variables. When one name is received, it is stored in the corresponding place of the array for the names. The addresses of the names of the variables are saved in two arrays in the PLC, one for the reals, one for the booleans. So after responding each request, it is accessed the next name.

Now, the records are required with another FOR loop. When a request is received in the server, it is used a function block from the remote library, *ReadArchive()*, which is used to read all the records in the archive (with each execution is obtained next record). The date&time value in this record, is transformed in a string variable, in a suitable format so it will be able to be transformed to a date&time variable later in the client, and now it is written in the buffer with all the values of the record. When the reponse is received by the client, each part of the record (date&time, real values, and boolean values), are stored in its respective part of its array.

To relate the records and the names, the first name in the array of names, is connected to the first variable in the array of reals, the second name with the second real, and so on. For example, if we have 10 variables, 3 reals and 7 booleans, the name of the fourth variable in the array corresponds to the variable placed in the first row of the array of boolean variables. Regarding date & time variables(1-D array), in the first position we have date and time for the first record, in the second position for the second record, and so on. It works like the columns in the 2-D arrays for the values, first column with first record, second....In the following figure, it is shown first the array for the names, and then the arrays for real and boolean values.

1	2	3	4	5	6	7	8	9	10

1									
2									
3									

4									
5									
6									
7									
8									

9									
10									

Fig 42. Connection between different arrays

The process of how to manage the data existing in the arrays in the PC application is explained in a later section.

If the selected mode is in this case to monitor the value of the variables, a request to get list of variables to monitor is sent (fig. 29), then the reponse to this request (fig. 30) is received, and with the number of real and boolean variables to monitor it is fixed the lenght of two arrays, one to store a string with all the values of the variables to monitor online, and another array to store the name of the names of the variables to monitor. In this case the it is not necessary to lock the archiving process, it is not critical to read the current values of the variables while the system is running.

The are two buttons in the application to start running two different timers, one to get the value of all the variables available to be monitored, and another to monitor one specific variable. In the method to get the value of all the variables, a request (fig. 40) is sent to the server, and then the PLC create a reponse with the current time in a string format suitable to be converted in a date&time variable in the client, the rest of the values are also placed in the reponse. Once the client receives the data from the server, it is save in a row in the datatable. Otherwise, when one specific variable is required to be monitor, it is necessary to obtain its ID before sending the request (fig. 25 and 27). This is done by means of the position in the listBox placed in the program, and thus, in the arrays. So for example, if we have 10 variables to monitor, 5 real and 5 boolean variables, they will be placed in the right order in the listBox, therefore, it will be easy to associate which is the right position in the array when the user selects one variable in the list.

It is also interesting to note that it will be activated a timer to send a request every 2 seconds to check if the connection is still working (fig. 38), in case no reponse is receive from the server or a wrong reponse, a variable is incremented, when this variable has been incremented two consecutive cycles, the socket will be disconnected.

11.- PC application

A Windows application is created to work as user interface. This application is created using the program *Microsoft Visual C# 2010*. The user will be able to connect to a PLC just specifying the IP and the Port where it is connected the PLC.

First of all, a window to connect to the system appears, so the user of the application can specify the IP and the port where the PLC is connected. An image of the appearance of this window is shown below:

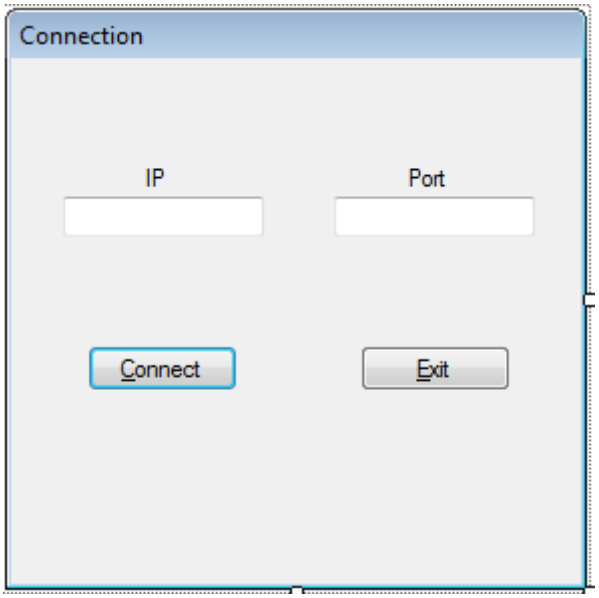


Fig 43. Window to connect the system

To achieve this, it is necessary to add a new *Windows Forms*, where there are two buttons and two text boxes as it is shown in the fig. 30. There is one bool method (the value returned is a bool) called *connection()*, where all this mechanism is carried out. Here the code is quite simple, there is a global object created from the class *Socket*. The socket object is filled with the data from the textboxes, using first the method *TryParse()*, so with this method, it is checked first if the values specified can be converted to IP or to int16 (for the port), so if they are not valid values it will appear a *MessageBox* to warn the user that he parameters are not correct. Otherwise a message to explain the connection was properly carried out or not will also appear. In this point, the structure *try* and *catch* is used. So if the connection it is not possible, there is an exception, and the code existing in *catch* is executed. If after 3 times trying to connect, it was not possible, the method returns a false value.

After the system has been connected correctly, a new window appears, and this window is to select the functional mode the user wants to work with. These two functional modes are, *Online monitor* and *Whole archive*.

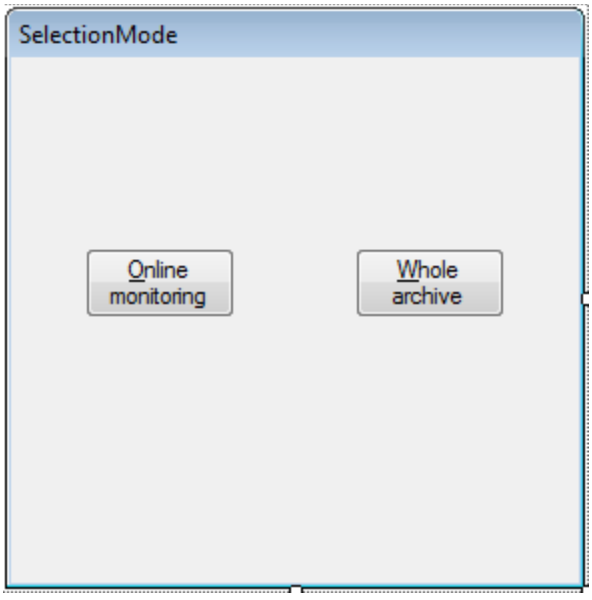


Fig 44. Window to select the mode

This window is programmed in a similar way than the window to connect to the system. This way, it is necessary to add a new *Windows Forms*, this window is used in a bool method called *SelectionMode()*, the algorithm inside this method is quite simple, so the only thing necessary is to return a true value in case the online monitoring button has been pressed or a false value in case the whole archive button has been pressed.

There are a group of variables and objects declared in a class *globalvariables*, it is important to have access to them in all the program. These variables are *static*, so the last value of this variable remains until next call, they are also declared as *public*, so they can be accessed in all the program. These variables and objects are shown in the table below :

Name	Data type	Description
server	*Socket (class)*	To connect and disconnect from the server
msg	1-D byte array	To create the request the reponses
NamesWholeA	1-D string array	Store the names of the vars for the WA mode
NamesOM	1-D string array	Store the names of the vars for the OM mode
ValuesR	2-D float array	Store the values of the real values WA mode
ValuesB	2-D boolean array	Store the values of the bool values WA mode
DateandTime	1-D D&T array	Store date and time variables for WA mode
DateandTimeX	1-D Xdate array	Store d&t variables in a special format
NReals	int16	Number of real values in the archive
NBooleans	int16	Number of boolean values in the archive
NRecords	int16	Number of records in the archive
ValuesOM	String array	All the values monitored received in a reponse
NrealsM	Int16	Number of real values to monitor
NboolsM	Int16	Number of boolean values to monitor
Activerow	Int	Number of the row active in the datagrid (WA)
Activerow2	Int	Number of the row active in the listBox (OM)

Table 20. Variables and objects contained in the *globlalvariables* class

Now, if the user has selected the mode *Whole archive*, the archive is required and received how it was explained in a previous paragraph (see at the end of 10.- Communications frames used).

When the real and boolean values, the names of the variables and the date and time values are in their respective arrays, they are represented in a data grid. To achieve this, a datatable is created, and then is filled with the different records from the archive using some FOR loops. Finally this datatable is used as source of the datagrid. An overview of how the interface will look is as follows:

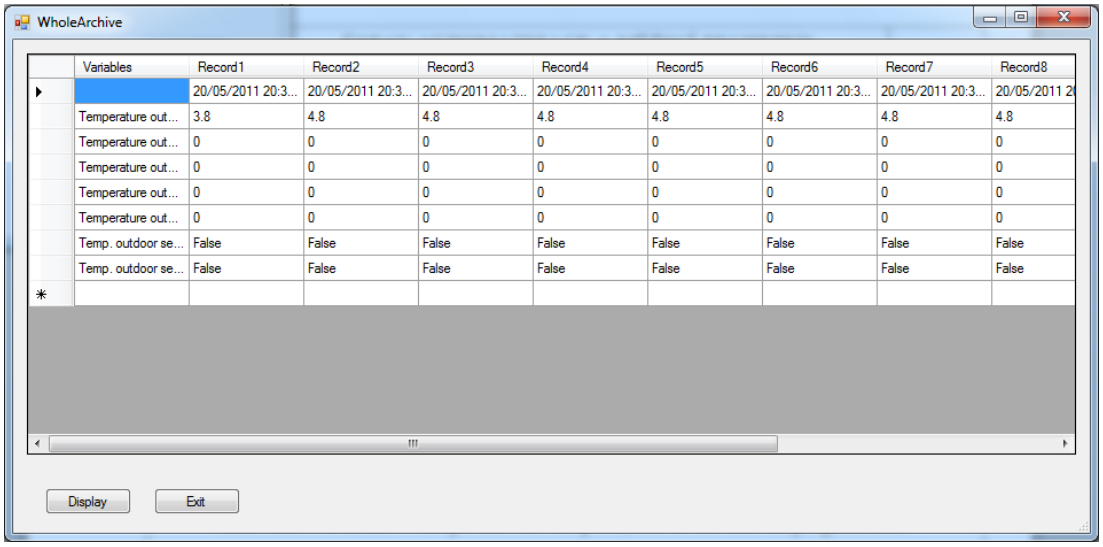


Fig 45. Overview of the whole archive mode

Besides the datagrid there is also a chart and a button. These two elements are to represent data from the datagrid, so when there is a cell selected or a row, and the button to display data is pressed, the data is represented in the chart as a line. This is achieved with the property *CurrentRow* of the datagrid, so selected row number can be got, and then use that number to represent a data from the arrays. To achieve this, a control from the web www.codeproject.com has been used. It is possible to create different charts for different variables and thus compare them.

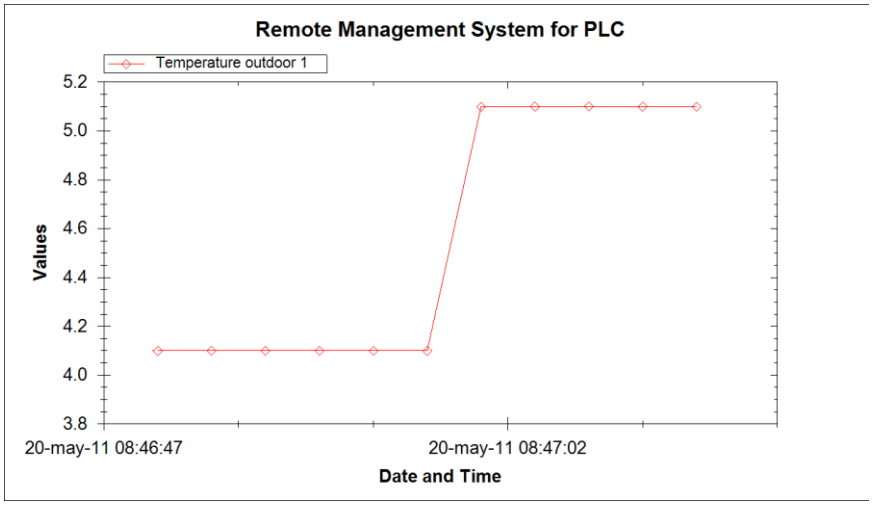


Fig 46. Chart of a variable

This control has the functionality of zoom, really useful in this case, because this system is intended to be used in large periods of time, and with a lot of records in the archive.

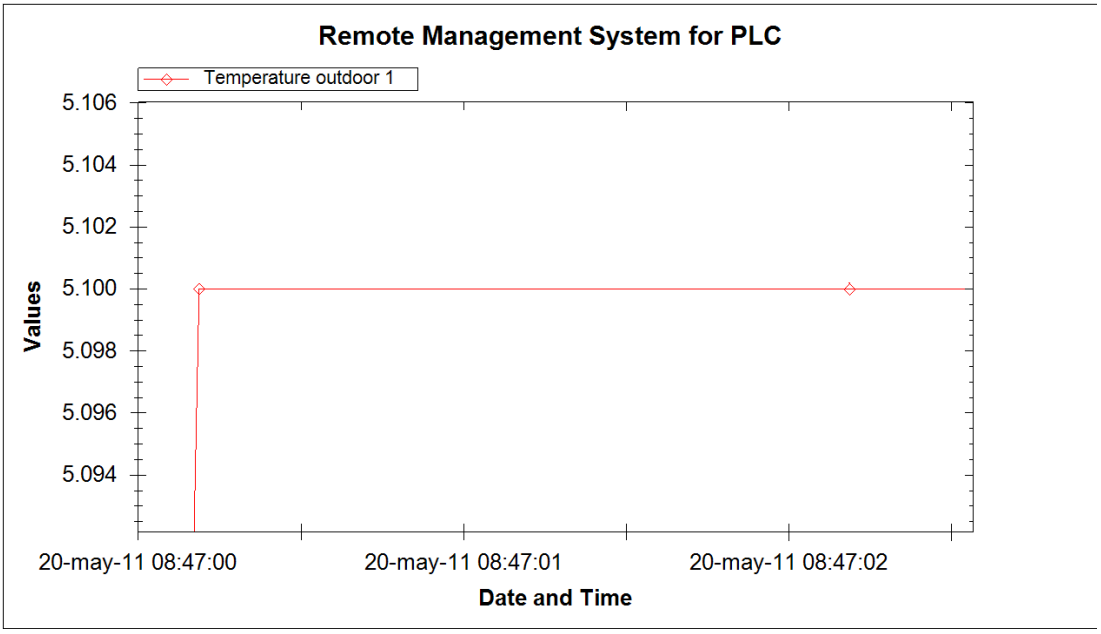


Fig 47. Chart with zoom applied

Now, when the user of the application selects the online monitor mode, it will appear a windows like the following:

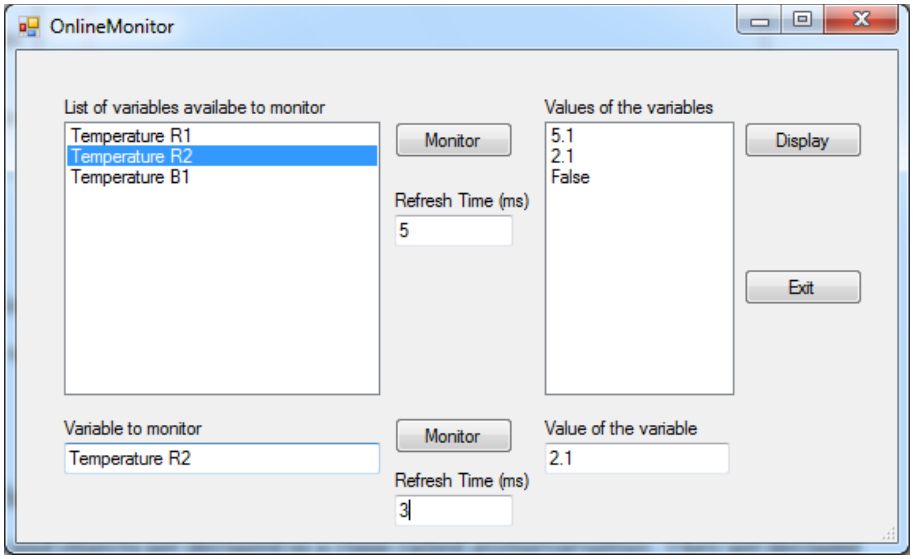


Fig 48. Overview of the online monitor mode

In the listBox from the top, there are all the variables that the programmer connected to the *Online_monitor* block. When this window is launched, the names of these variables are required with a FOR loop, and then they are stored in the proper array and written in the listBox. In the mean time, it is created one column per variable in the datatable created in the *globalvariables*. Now user is able to select the refresh time to monitor these variables, so a timer is then activated and it will require the values for the variables, when these values are received, they are written in a new row in the datatable. Note that the datatable can store up to 16777216 rows, it means the variables can be monitored during more than 194 days each second.

12.- Brief end-user instructions

PLC library

The following sets out a simple series of instructions for using the library:

- One variable type Control has to be declared as global variable.
- There are some required libraries which have functions or functions blocks used in the library. These libraries must be added to the project, apart from the created library, *Convert*, *Runtime*, *Operator*, *asstring*, *astime*, *brsystem*, *DataObj*, *standard*, *sys_lib*, *AsTCP*, *AsPciStr*(SG4) and FileIO(SG4).
- It is considered elemental, but the programmer must create some variables for the inputs, and for the functions blocks used. I.e. one variable type ArchReal() per each real input of the system.
- When the parameters for the function block Engine() are assigned, beware with the values for the cyclic period and the archiving period, because the value for the archiving period has to be equal or a multiple. Something similar happens with the value assigned for the input which specifies the size of the data objects, it has to be smaller than the value specified for the archiving.
- When the function block monitor online the values, the datatype has to be specified, so a 0 is to specify it is a real value, and 1 is to specify is a boolean value.
- All the instructions will be in the cyclic part, so the init part will be empty.

  After all, an example of the cyclic part for a system with 3 real inputs, and 2 boolean inputs is as follows (next page):

```
Real1.Control:=ADR(ArchControl);          (*The parameters for the first real input*)
Real1.RealIn:=ADR(Temperature1R);
Real1.Name:='Temperature outdoor 1';
Real1();
Real2.Control:=ADR(ArchControl);          (*The parameters for the second real input*)
Real2.RealIn:=ADR(Temperature2R);
Real2.Name:='Temperature outdoor 2';
Real2();
Bool1.Control:=ADR(ArchControl);          (*The parameters for the boolean input*)
Bool1.BoolIn:=ADR(Temperature1B);
Bool1.Name:='Temp. outdoor sensor 1';
Bool1();

Engine1.Control:=ADR(ArchControl);        (*The block Engine is configured before it is used*)
Engine1.MaxSize_Bytes:=700;
Engine1.SizeDObj_Bytes:=120;
Engine1.CyclicPeriod:=1000;
Engine1.ArchPeriodms:=1000;
Engine1();

RecRead1.Control:=ADR(ArchControl);       (*The block to read specific value in a record is configured*)
RecRead1.Valuespecified:=5;
RecRead1.Name:='Temperature outdoor 1';
RecRead1();

Online1.Control:=ADR(ArchControl);        (*To monitor one variable*)
Online1.Input:=ADR(Temperature1R);
Online1.Name:='Temperature R1';
Online1.Data_type:=0;                     (*0 to specify real value*)
Online1();

Online2.Control:=ADR(ArchControl);
Online2.Input:=ADR(Temperature2R);
Online2.Name:='Temperature R2';
Online2.Data_type:=0;
Online2();

Online3.Control:=ADR(ArchControl);
Online3.Input:=ADR(Temperature1B);
Online3.Name:='Temperature B1';
Online3.Data_type:=1;                     (*1 to specify boolean value*)
Online3();

Server1.Control:=ADR(ArchControl);        (*Block to configure the PLC as a server*)
Server1.PortNumber:=12000;
Server1();

ArchInfo1.Control:=ADR(ArchControl);
ArchInfo1();
```

Figure 49. Cyclic section of an example program

The order is not important, so for example, first can be placed the *Engine()* and then the blocks to archive the real values or boolean values, or first can be placed the blocks to archive the values and then *Engine()*.

PC application

The PC application is really easy to use for the user, the application guides the user through the process in a friendly way. When the application is executed, a window like the following appears:

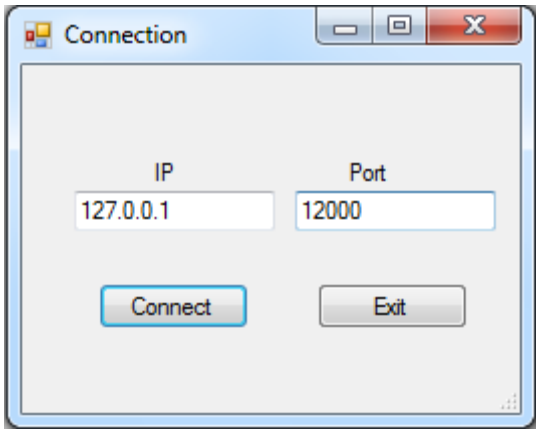


Figure 50. Window to connect to the PLC

Here, the user has to insert the IP address type 4, for example 127.0.0.1 and the number of the port where it has to connect. If the parameters have been correctly inserted and the connection was possible, a boxdialog is shown:

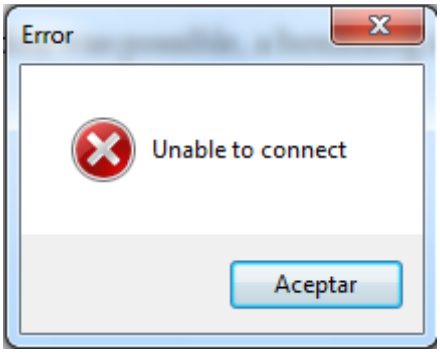


Figure 51. boxDialog showing an error

Otherwise, it will be shown one like this:

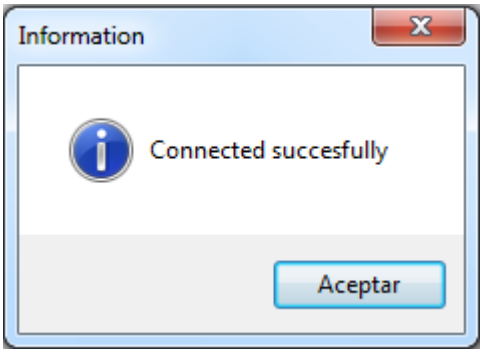


Figure 52. textBox after a succesful connection

In case the parameters are not correct, the user is also warned, showing a box dialog like this:

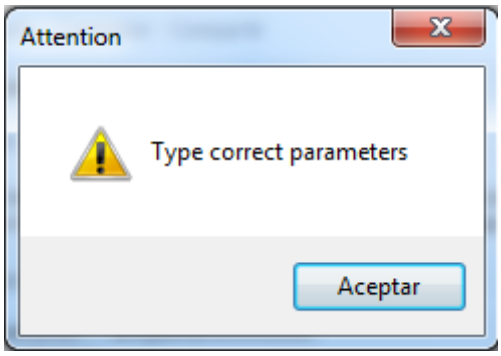


Figure 53. textBox to warn the parameters are not correct

When the client has been correctly connected to the server , a window to select what kind of operation mode the user wants to run appears:

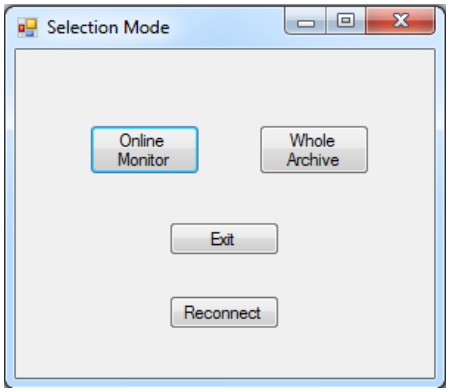


Figure 54. Window to select to operation mode

When the system is the system has som failure due to a lost connection event, it will automatically appear a box dialog like the following:

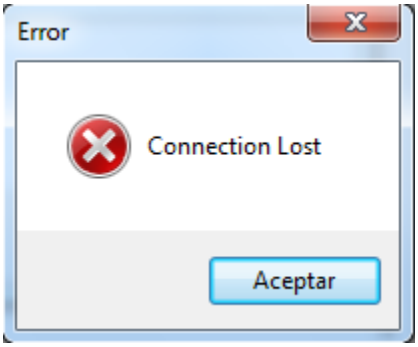


Figure 55. Window to inform the connection is lost

After this, the window is closed, and the selection mode window appears again, where it is possible to try to connect again with the server.

13.- Tests and conclusions

Automatical archiving and communication with the PC application

To test this part of the system is going to be used a CPU type CP0292 connected to X20BB27 power distribution backplate and X20PS9500 interface. The main features of this CPU are the following:

X20 CPU,
Compact CPU µP 25,
750 KB SRAM, 3 MB FlashPROM,
RS232 and CAN support
corresponds to Compact CPU base,
1 Ethernet interface 100 Base-T,
order bus base, supply module and
terminal block separately.

Figure 56. PLC used to test the automatical archiving

The test which will be carried out will be a system to archive the values of 5 real inputs and 5 boolean inputs. The parameters set in the *Engine()* block are 700 bytes for the maximum size reserved for the archive, 120 for the maximum size of one data object, 50 ms for the cyclic period of the PLC and 1000 ms is when one record has to take place.

After some errors at the beginning, due to the fact that the dynamic arrays created for the addresses of the inputs and the ID's of the data objects, it start working properly.

The system is left working for 10 minutes, and the results appear satisfactory, regarding the global variable type control, the size of one record is 29 bytes, 6 usual data objects, no smaller data object (no space for 1 record), 24 records when the system is working in overwriting mode, 6 data objects and 4 records per object.

When the system tries to connect, the connection is carried out properly, and the system seems to start working properly. However when the exchange of bytes starts, there are some errors, due to the fact that the names and the values of the variables in both operation modes are not received correctly. After debugging and analyzing in detail the system, it can be seen that the error comes from the function *memmove()*. This is the main function used in the *ServerCreate()* to modify the data buffer to send to the client, it works in a different way in the PLC and in the simulator.

Conclusions

After this study, it can be concluded that the system maybe is almost ready to be implemented in a professional application, but it would be required first some more exhaustive test and analisys to check that the functionality is correct. Besides of this, there are two interesting extensions which could be worked in the future, these are the following:

1. Carry out more testing experiments and debug the program.

2. Study the possibility of creating the global variable (*Control* type from remote library) in remanent memory, so in case of a power loss, the archiving process could start from the same point.

3. Improve the charting or find another control, so it could be represented in the same graph as many variables as the user wants.

Now it is possible to represent several variables and compare them but they are in different windows.

4. Test the functions to save the archive in a CSV file in a USB stick.

Bibliography

The information for creating this thesis has been obtained from:

[1] Wikipedia contributors, *Transmission Control Protocol*, Wikipedia, available from http://en.wikipedia.org/wiki/Transmission_Control_Protocol, last modified 26 December 2010.

[2] Wikipedia contributors, *User Datagram Protocol*, Wikipedia, available from http://en.wikipedia.org/wiki/User_Datagram_Protocol, last modified 17 December 2010.

[3] Hasan Hyder, *Modbus RTU Overview*, Real Time Automation, Inc, available from http://www.rtaautomation.com/modbusrtu/

[4] John Rinaldi, *Modbus TCP Overview*, Intellicom, Inc, available from http://www.intellicom.se/solutions_ModbusTCP_overview.cfm

[5] Corporate Headquarters, Bernecker + Rainer Industrie-Elektronik Ges.m.b.H., TM210TRE.30-ENG (Training Module), *Automation Studio Basis.*

[6] Corporate Headquarters, Bernecker + Rainer Industrie-Elektronik Ges.m.b.H., TM250TRE.00-ENG (Training Module), *Memory Management and Data Storage.*.

[7] Wikipedia contributors, *Ethernet/IP*, Wikipedia, available from http://en.wikipedia.org/wiki/EtherNet/IP, 3 December 2010 at 15:00.

[8] Andy Swales, Schneider Electric, *Open Modbus/TCP Specification*, Release 1.0, 29 March 1999.

[9] B&R automation company, official website, available from http://*www.br-automation.com*

[10] John Rinaldi, *EtherNet/IP Overview*, Real Time Automation, Inc, available from http://www.rtaautomation.com/ethernetip/

[11] Unknown authors, *Visual C# .NET Programming*, Home&Learn, available from http://www.homeandlearn.co.uk/csharp/csharp.html#

[12] JChampion (Nickname), *A flexible charting library for .NET*, The code project, available from http://www.codeproject.com/KB/graphics/zedgraph.aspx

[13] Microsoft staff, *MSDN help*, Microsoft, available from http://msdn.microsoft.com/es-es/ms348103