

Ada for Automation

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
2015.02	2015-02-15		SLO

Contents

1	Introduction	1
1.1	Automation and controllers	1
1.2	Fieldbuses	1
1.3	Industrial PCs	2
1.4	Control engineers	3
1.5	Languages in automation	3
2	Ada for Automation	5
2.1	What is it?	5
2.2	Goals	5
2.2.1	A powerful language	6
2.2.2	A profitable investment	6
2.2.3	Quality recognized but ignored	6
2.3	For which project types ?	6
2.4	In conclusion	7
2.5	License	8
2.5.1	Source code	8
2.5.2	This document	8
2.6	Obtain Ada for Automation	9
2.7	Contribute	9
2.8	On the Web	9
3	About Ada	10
3.1	French tutorial	10
3.2	Development studio	10
3.3	Support and training	10
3.4	Some references	11

4	Concepts	12
4.1	Security	12
4.2	Real Time	12
4.3	Structuration	12
4.3.1	Tasks	12
4.3.2	Sections and organization blocks	13
4.3.3	Subroutines	13
4.3.4	Functions	13
4.3.5	Function blocks	13
4.3.6	Libraries	13
5	Getting started	14
5.1	Register on the forum!	14
5.2	Acclimatization	14
5.3	Get started!	14
6	Guided tour	16
6.1	What is available today	16
6.1.1	libmodbus	16
6.1.2	Hilscher	17
6.2	Directories	17
6.3	GNAT Pro Studio Projects	18
7	Design	21
7.1	General Architecture	21
7.2	Applications	22
7.2.1	Console application	22
7.2.2	Application with Graphical User Interface	23
7.2.2.1	Identity tab	26
7.2.2.2	General Status tab	28
7.2.2.3	Modbus TCP Server status tab	30
7.2.2.4	Modbus TCP Clients status tab	32
7.2.2.5	Modbus RTU Master status tab	34
7.2.2.6	Hilscher cifX status tab	36
7.3	Packages	38
7.4	Tasks	38
7.4.1	Main_Task	38
7.4.2	Periodic 1	39
7.4.3	Sig_Handler	40
7.4.4	Clock_Handler	40

7.5	Modbus TCP IO Scanning	40
7.6	Modbus TCP Server	43
7.7	Modbus RTU IO Scanning	44
7.8	Data Flow	45
7.9	Interface with the user program	47
8	Application Example 1	48
8.1	Functional specifications	48
8.1.1	Object	48
8.1.2	Description	49
8.1.3	Operating Modes	49
8.1.3.1	Manual Mode	49
8.1.3.2	Automatic Mode	49
8.1.4	Human - Machine Interface	50
8.1.5	Control panel	50
8.1.6	Instrumentation	50
8.1.7	Inputs / Outputs	50
8.1.7.1	Analog Inputs	50
8.1.7.2	Digital Inputs	50
8.1.7.3	Digital Outputs	50
8.1.8	Human - Machine Interface	50
8.1.8.1	HMI \Rightarrow Control	50
8.1.8.2	Control \Rightarrow HMI	51
8.1.9	Simulation	51
8.2	Overview	51
8.3	Organic specifications	51
8.3.1	Assignment of Inputs / Outputs	52
8.4	The app1 project	52
8.5	Memory areas	53
8.6	Modbus TCP IO Scanning Configuration	54
8.7	Modbus TCP Server Configuration	55
8.8	User objects	56
8.9	I/O Mapping	59
8.10	Main procedure	61
8.11	Kernel loop	64

9 Hilscher	66
9.1 Components	66
9.1.1 The netX family	66
9.1.2 The rcX Real Time Operating System	66
9.1.3 Hardware abstraction layers	66
9.1.4 Protocol stacks	67
9.2 Products designed around the netX	67
9.3 Configuration tools	67
9.4 Drivers	67
10 Application Example 2	68
11 Application Example 3	69
12 Library	70
12.1 Basic types	70
12.1.1 Types	70
12.1.2 Shifting and rotation	71
12.1.3 Bytes and words arrays	72
12.1.4 Text_IO	72
12.1.5 Unchecked conversions	73
12.2 Conversions	73
12.3 Analog processing	74
12.3.1 Scale	74
12.3.2 Limits	75
12.3.3 Ramp	75
12.3.4 PID	75
12.3.5 Thresholds	76
12.4 Timers	76
12.4.1 TON	76
12.4.2 TOFF	77
12.4.3 TPULSE	77
12.5 Components	78
12.5.1 Device	78
12.5.2 Alarm Switch	78
12.5.3 Contactor	78
12.5.4 Valve	79
13 Bibliography	80
13.1 Books	80
14 Glossary	81

To my family and friends.

To the little white horses of Mister Brassens.

Chapter 1

Introduction

"Ada for Automation" (A4A in short) is a framework for designing industrial automation applications using the Ada language. It makes use of the libmodbus library to allow building a Modbus TCP client or server, or a Modbus RTU master. It can also use Hilscher communication boards allowing to communicate on field buses like AS-Interface, CANopen, CC-Link, DeviceNet, PROFIBUS, EtherCAT, Ethernet/IP, Modbus TCP, PROFINET, Sercos III, POWERLINK, or VARAN.

It is necessary to clarify a little the context, which will be done briefly thereafter.

1.1 Automation and controllers

Historically, automated systems are managed since the 1980s and the advent of microprocessors by specialized computers which are designated by the term Programmable Logic Controller or PLC.

The system to control, a machine, a chemical reactor, biological or nuclear, any craft, has sensors that allow to know at every moment the state of the system, and actuators that enable to act on it.

The sensors provide binary information following Boolean logic, true or false, hot or cold, open or closed, etc. or information about a measured quantity, temperature, pressure, speed, Analog said, and in fact is a physical quantity converted into an electrical signal that is digitized in order to treat programmatically.

At that time, the controllers received their information or controlled their actuators with electronic cards arranged in a chassis that provided both a mechanical and electrical protection against electromagnetic interference.

The processing unit or CPU (Central Processing Unit) also lived in the same chassis or rack. One could extend the chassis via expansion flat cables and so have more inputs to acquire signals, and outputs to control actuators.

Each sensor's information is given by switching a 0 voltage - from 24 to 48 or even 110 volts for the digital inputs, or by generating a signal 0 - 10V for a voltage input, 0 - 20mA or 4 - 20mA current input.

The actuators are controlled either binary, via relay cards, power transistors, triacs ... or by sending an analog signal 0 - 10V, 0 - 20mA or 4 - 20mA to a regulator, a dimmer, a drive...

On a large installation, the controller chassis occupied a prominent place in the control cabinet and the amount of cables needed was proportional to the amount of connected sensors and actuators. For a sensor, a cable for the signal and a power cable, which could supply several sensors optionally, were necessary. For actuators, as it is possible to have a cable for analog signals, say the speed to be specific, a cable for digital signals, power / stop control as example and of course the power cables, cables installers had work.

All these cables have a cost, to purchase, in studies, implementation, and maintenance, knowing that the connectivity problems represent about 80% of the failures.

1.2 Fieldbuses

In the 1990s, it began to appear as fieldbuses Modbus or PROFIBUS.

A fieldbus is a cable, it is said a medium when it is a specialist, which runs from equipment to equipment, and which operates a communication between these devices according to a communication protocol.

As there are always multiple solutions to a problem, it has established a number of protocols, a new Babel.

On this fieldbus can be encoded signals from multiple characteristics to achieve transfer those signals with a good performance and an ever lower cost, performance agreeing by various criteria such as flow, speed, resistance to disturbances, distance, safety ... Every compromise reached has its supporters and detractors.

To develop, standardize and promote each of these technologies, organizations are put in places.

On this fieldbus therefore, new communicating devices are emerging, from modules with inputs and outputs allowing local deport of the acquisition and control of sensors and actuators, but also drives for controlling motors, dimmers, all kinds.

Supply and fieldbus are enough to control installations by PLCs whose processing capacity increase accordingly.

Always looking for higher performance and cost reduction, the industry created since new protocols based on an affordable and efficient technology, the Ethernet technology. Of course, there is always a downside, and it took a lot amend this technology before it is suitable for process control applications.

As there are always multiple solutions to a problem, it has established a large number of new protocols, a new Babel ...

These communication abilities are also benefiting to monitoring and diagnostic tools that can refresh the data faster and more transparent. They enable better availability of facilities and greater control.

One could mention the following advantages:

- it is possible to insert on a fieldbus equipments from different manufacturers, which is less obvious in PLC racks. The architect of the solution is thus more free to choose the elements.
- the modularity of remote I / O stations is much higher than that of cards in racks. It is possible to mix the inputs / outputs of different types in a station while a card has a function (input or output - digital or analog) with a fixed number of channels.

1.3 Industrial PCs

Mainframe computers were found since long to manage complex processes, blast furnaces, refineries ... With proprietary operating systems, engineers for their administration and others for ongoing maintenance, these systems were not for everyone.

PCs invaded the offices but were confined to supervision as judged not reliable enough for the control and monitoring. There were many attempts but had bad press. They are reserved to the specific needs that require the processing power not available with processors powering the controllers or storage capacity or access to data in a database.

Since then, things have changed a bit, Linux has been there and reliability of operating systems has improved. The devices are also much cheaper and reliability is more important when choosing configurations without rotating parts.

The Mean Time Between Failure (MTBF) of an industrial PC is very close to that of a PLC, that said, it's quite normal since they share many components today.

And providing scalability and openness incommensurate with the PLC, industrial PC has indisputable advantages.

When we needed a PC for supervision or advanced treatments, some have questioned why we would not do control-command with the computing power available, leaving the PLC.

We gain in cost, compactness, simplicity in architecture ...

So we added some real-time extensions to general operating systems and developed automation solutions for the PC.

Siemens with WinAC® RTX® and real-time extensions from IntervalZero®, ISaGRAF®, CoDeSys®, ProConOS®, Straton®, TwinCAT®... Many.

It is possible to install I / O cards in the PC and thus obtain an architecture strongly resembling a rack controller. It is even the most widely used in many demanding applications solution.

It is also possible to install in the PC one or more industrial communication cards, such as those of the Hilscher company for example, to provide a connection to the fieldbus in master mode to control equipments or in slave mode to communicate with the upper level.

Control engineers are still somewhat reluctant to mix computers and automation. But times change, seniors are classed as oldfashioned by juniors grown up with Web 2.0, technologies are pushing the wheel and there is a convergence PC - PLC with an economic situation that compresses the budgets ...

1.4 Control engineers

A control engineer or PLC programmer is a specialist in automation. It is asy to say, but it is a difficult task to explain what that means.

Depending on the area, chemical, petrochemical, pharmaceutical, manufacturing, building management, special machines ... , his initial training, instrumentalist, electrician, mechanic, electrician, maintenance technician, the structure of the company that employs him, his curiosity and his inclinations finally, the standard automaticien profile is rather vague and extremely varied are his tasks.

It may be necessary for him to perform alternately several occupations on the same project. It is absolutely not unusual for a control engineer to carry out studies on instrumentation, electrical, automation and supervision, including networks and databases, etc ...

Note that the computer science is not mentioned in his abilities. This is actually on the job and often a bit of obligation he learns it.

So a control engineer designs PLC programs, among other things but often has only some culture, and not a training, in computeur science.

1.5 Languages in automation

Also, he has been concocted a range of languages tailored to his tasks, process of combinatorial logic, state machines, control or regulate quantities, monitor processes ...

Thus we find:

- the Ladder language with its contacts in serie or parallel to symbolize the AND and OR, relay coils to store states, the language of electricians, automation has replaced the tens of meters of relay cabinets of yesteryear.
- logic diagrams, a representation designed for electronics with their AND, NAND, NOR gates ... little used today it seems.
- the instruction list, a pseudo assembler for electronics in the digital era, before the advent of C.
- the Sequential Function Chart, a French (GRAFCET) specialty for the representation of state machines suitable for mechanics, when the mechanics are slow ...
- the block diagrams, where the system is represented by functions blocks connected through wires, adapted to control, servo ...
- the structured text, an ersatz of Pascal, Ada, C ... supposed to limit "creativity" and bugs that go with.

All these languages have integrated debugging capabilities and dynamic visualization, which is very convenient, we must admit.

Moreover, it is usually possible to make online changes to the program, ie during operation of the machine without having to reset it, which is very useful when starting the installation. Of course, there are some caveats and some operations require recompiling the entire program, such as configuration changes, those relating to the interface of functions and blocks, etc ...

However, these languages are adapted to the tasks of limited scope and their expressiveness is bad enough.

Once a Boolean equation is somewhat complex, the ladder networks become difficult to understand and it is even more difficult to develop them without introducing regression. The programs are miles long.

The logic diagram is gone, we have not heard a complaint.

Instruction list or LIST, pseudo assembler is as austere as the real assembler. Difficult to write, even more difficult to read it. The preferred language of my freelance colleague who said, "we must spoof the client". He succeeded the guy ...

SFC or GRAFCET... So, especially in France.

With all these languages, that is if one is trained in their practice and their characteristics, we get to do things, especially if one chooses the most appropriate language.

Should they still be available. This is not the case in general, salesman gratify us with their drawer offers, and managers often have a short-termist view. Thus, such a workshop would propose that basic ladder, the logic diagram - useless, and instruction list.

As a smart manager, one will not, for example, invest in more advanced languages such as structured text or block diagrams because they offer higher productivity in many cases. Thus, the PLC programmers, who will not be trained and will not be able to train themselves because lacking the tools, will continue to develop in ladder or assembly language.

Over time, these same PLC programmer end up working as customers -the dream of every developer in software company- and impose their specifications the contact language or assembler. No functions or function blocks because it's too complicated ... It's been lived.

In another development environment, we will not have the ability to write a simple function ... Exasperating. We hear about object-oriented programming and we are not able to do only structured programming!

There are also systems in which more or less high level languages are used as C / C ++, Basic, Delphi and others. It's pretty confidential but it is.

Visual Basic for Application that can be found in Microsoft Office® applications or in PcVue® supervision from Arc Informatique is also practiced.

Chapter 2

Ada for Automation

So there has been a multitude of proprietary solutions for the automation on a PC base. That is not shaking the PLC market. Old habits die hard, and manufacturers are prudent people, sometimes conservative . . .

Software has done its revolution with the Free Software. Above all else, the contribution that we find essential is almost philosophical: replace the competitive ideology by the contributory one. Helping each other, it's better than killing each other. One can argue his best, there is no argument that can face this fact.

2.1 What is it?

"Ada for Automation", so this is a framework, ie open source code to allow you to write more easily and quickly own applications. With some success, this framework is expected to grow with time and contributions.

This application framework provides an environment where there are a number of elements such as execution loops, communication means to the inputs and outputs, but also to monitoring tools, libraries of functions and components . . .

It is necessary to add your application code to get something useful.

You will need to learn Ada, slowly but surely, to use it. However, the availability of source code will help you and if you know the ST language for "Structured Text" you will be surprised by a certain familiarity.

Ada is a compiled language, you will have to learn how to use the tools and debug techniques. Ada is a particularly powerful and elegant language, while remaining relatively easy to learn for a technician or an automation engineer.

En fait, celui-ci retrouvera dans ce langage les concepts qu'il manipule au quotidien en utilisant son atelier de programmation d'automate favori -ou imposé-, avec une syntaxe claire rappelant le langage "Structured Text" de la norme IEC 61131-3, ce qui est somme toute normal puisque ce langage s'inspire notamment de Ada. In fact, the PLC programmer will find in this language concepts he manipulates everyday using his favorite -or imposed- PLC programming workshop with a clear syntax reminiscent of the language "Structured Text" of the IEC 61131- 3, which is quite normal since this language draws particularly from Ada.

2.2 Goals

Some goals have been identified for this project.

- Have adequate language for demanding automation applications, but also suitable for basic applications, multiplatform.
 - Allow the PLC programmer to train himself on the paradigms (sub) used in current programming environments, which will make him more productive on these environments.
 - Allow Ada to get out of its elitist ghetto that its origins and history have locked it in, sensitive applications in military, nuclear, aerospace, finance, medical, rail . . .
-

2.2.1 A powerful language

While originally PLCs were replacing electrical cabinets in which logic was wired, today's controllers communicate via network with their peers, supervisory tools, web, databases, MES (Manufacturing Execution System) ... The new concept "fashionable" Industry 4.0, should accelerate this general trend: innovation through functional integration.

The reusable component concept has become central because precisely it helps to capitalize developments and facilitate the reuse. Ada, with packages for organizing modularity and object programming, provides since its inception in 1983 the means to create these components.

Applications are now distributed, as is intelligence, and communication unites all these autonomous entities.

This concept finds its realization in standards such as IEC 61499 implemented in ISaGRAF ® or Siemens with its "Component Based Automation". Ada with Annex "Distributed Systems Annex" is no exception.

Ada is a general-oriented language. It is possible to write in this language all kinds of application. "Ada for Automation" does not limit this vocation while facilitating its adoption by a community closer to the ground.

2.2.2 A profitable investment

The study of this language is fascinating. Ada is a monument to the glory of Software Engineering. All concepts developed over years of development of this science are reflected in Ada. This makes this language the witness of the evolution of this science and the crystallization of it.

The language is designed to create both critical applications in embedded and real-time for equally critical applications for large systems. Modularity, encapsulation, structured programming, object oriented programming, concurrent programming, multitasking, real-time distributed processing ... And it's all free!

With the availability of development tools, documentation and source code, the learner has all the cards to evolve according to his needs and according to his desires.

In general, one becomes PLC programmer by chance and often in pain. He learns on the job, often not in good conditions, with the pressure of the result, the client and the hierarchy.

In fact, apart from the subject matter, information, the fundamentals are the same. Designing an automation program is like designing a computer program. And the same mistakes lead to the same disastrous results.

Ada promotes good design practices: strong typing, modularity, encapsulation, objects and reusable components ... By learning and using Ada "Ada for Automation" on some projects, an automation engineer can probably gain some teaching.

2.2.3 Quality recognized but ignored

Attempts to explain why such a virtuous language Ada remained so confidential can be found on the web. Perhaps this is due to areas where it is traditionally used, unwilling to communicate. Many other reasons are given. Maybe there just is not enough usage examples, that is what "Ada for Automation" want to help change.

Perhaps also is Ada impressive. The amount of concepts expressed in this language is simply exceptional and books are thick and often available only in English. Other languages seem more affordable and quickly controlled. This may be the case, but every coin has two sides: what about when the application becomes large, was written long ago by today absentees and unavailable tools?

2.3 For which project types ?

Ada is a compiled language, with a performance comparable to that of the C / C ++. On current platforms, the performance obtained allows to consider the migration of a large majority of applications today running on controller.

Does this mean that "Ada for Automation" is opposed to the PLC manufacturers? Of course not!

On the one hand, most PLC manufacturers also have an offer "PC based Automation". This is particularly the case with Siemens WinAC ® on MicroBox and Nanobox for example, but also from Beckhoff, B & R ...

Furthermore, in the event that "Ada for Automation" is adopted by a significant fringe of PLC programmers, nothing prevents PLC manufacturers to incorporate this possibility in their range. That's also that free software!

Depending on the used hardware platform and operating system, the system architecture, the distribution of treatments, all applications are possible.

- On entry-level processors and standard OSES such as Microsoft Windows ® or Linux / Linux RT can be considered Centralized Technical Management or Building Management System (BMS / BMS), Infrastructure Management, treatment of water and effluents, conveying and handling . . .
- On Windows platforms with RTX ®, VxWorks ® or Linux RT, Xenomai, RTEMS . . . and equipment suited one can access high-end applications such as special machines with motion control.

This may seem ambitious. The "Ada for Automation" project has for objective to provide an open platform for building automation applications. These are the users and contributors of the project, and of course the end customers, who will set their own limits.

However, this is achievable. The consum-actors are powerful entities that have an interest in seeing the establishment of standards, libraries of reusable components, good design practices.

One can also imagine a kind of vertical integration with "business" applications created by their users.

Other communities have been able to build in areas not so far away and got to shake the established monopolies.

Renovation or conversion of existing facilities are probably favorable phases in the evaluation / adoption of "Ada for Automation".

It is possible to save and reuse some, if not all, of the equipment and wiring equipping controller racks with fieldbus couplers in place of the central processing unit (CPU), I / O boards being maintained.

See for example the solutions developed by the company EFSYS: <http://www.efsyst.fr/>

Thus, only the control part is to overhaul, which must be done anyway regardless of the chosen solution. The compatibility of the old software with the new material is more than rare and even if that were the case, or if converters allow to approach it, it is probably desirable to disentangle changes made in the lifetime of the facility under suboptimal conditions.

Since renovating, let's do it right. We are working on the intelligence of the system.

2.4 In conclusion

"Ada for Automation" is an economical, powerful and durable solution for the realization of advanced control and monitoring applications.

Of course, we do not claim to revolutionize the field. The author is far from being an expert in Ada. It is even as a complete beginner he started developing this application framework.

And we know the resistance to change for having experienced them.

"Ada for Automation" is not intended to fans of a particular manufacturer / development environment / language. It is necessary to have some intellectual curiosity to overcome the acquired reflexes and venture off the beaten path. It will take some time for those interested to master the concepts and tools. But the concepts are universal and free the tools. We are certain that the benefits in terms of training and openness can be considerable.

We do not oppose automation languages and Ada. We think there is room for all. But we would be very happy if this language could find its place in the panoply of the automation engineer.

Ada would solve the problem of portability which is not yet resolved with PLCopen more probably due to lack of willingness of manufacturers than as a technical impossibility.

Maintainability has always been one of the major criteria in the development of Ada. The focus is on code readability because if it is written at least once it will be the subject of many proofreading.

It is obvious that the world of automation and industrial IT and those in which Ada is used traditionally share the same needs in terms of life expectancy, durability and reliability of the solutions.

With a standard written in 1983 and updated in 1995, 2005 and 2012 incorporating the latest in Software Engineering occurred in the meantime and considered interesting by experts from the fields of use and language, Ada shows a good health. And it is still possible with the latest tools generation to compile applications written to the origin of language, even though they would still deserve a little bit of dusting ...

We believe in the virtues of exemplary. If the fruits are good, is that the idea is good. This is the meaning, if not the formula ...

We believe that we may be limited for "Ada for Automation" to use the basic concepts of language, so as to be more intelligible to non-Adaïstes. This does not limit the use of Ada for the user application.

We have long been committed to the ideas of free software and hope to contribute to the revolution with "Ada for Automation".

Freedom is a right in principle ... but it must be deserved.

2.5 License

2.5.1 Source code

The source code of "Ada for Automation" is covered by the GNAT Modified General Public License (GMGPL) : http://en.wikipedia.org/wiki/GNAT_Modified_General_Public_License

Thus, this header is found in each of the following file:

```
--          Ada for Automation          --
--                                     --
--          Copyright (C) 2012-2014, Stephane LOS  --
--                                     --
-- This library is free software; you can redistribute it and/or --
-- modify it under the terms of the GNU General Public --
-- License as published by the Free Software Foundation; either --
-- version 2 of the License, or (at your option) any later version. --
--                                     --
-- This library is distributed in the hope that it will be useful, --
-- but WITHOUT ANY WARRANTY; without even the implied warranty of --
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU --
-- General Public License for more details. --
--                                     --
-- You should have received a copy of the GNU General Public --
-- License along with this library; if not, write to the --
-- Free Software Foundation, Inc., 59 Temple Place - Suite 330, --
-- Boston, MA 02111-1307, USA. --
--                                     --
-- As a special exception, if other files instantiate generics from --
-- this unit, or you link this unit with other files to produce an --
-- executable, this unit does not by itself cause the resulting --
-- executable to be covered by the GNU General Public License. This --
-- exception does not however invalidate any other reasons why the --
-- executable file might be covered by the GNU Public License. --
```

2.5.2 This document

Creative Commons License

The document you are reading is available under the terms of the [Creative Commons Attribution-ShareAlike 3.0 Unported License](https://creativecommons.org/licenses/by-sa/3.0/)

2.6 Obtain Ada for Automation

"Ada for Automation" is hosted by Gitorious :

<https://gitorious.org/ada-for-automation>

You can now obtain "Ada for Automation" with the usual git tools:

```
git clone git://gitorious.org/ada-for-automation/ada-for-automation.git A4A
```

or download an archive:

<https://gitorious.org/ada-for-automation/ada-for-automation/archive-tarball/master>

A Git tutorial is available here for example :

<http://www.kernel.org/pub/software/scm/git/docs/gittutorial.html>

2.7 Contribute

It is of course possible to contribute to this project:

- your comments, criticisms and suggestions are welcome and will be considered and debated.
- your library items, components, functions, etc. will enrich it.
- translate the documentation, this book in particular, will increase its distribution.
- users will benefit from your help on the forum.

A big thank you to all our contributors and in particular:

Aurélien MARTEL for the "Ada for Automation" logo

François FABIEN for his advice (francois_fabien@hotmail.com)

2.8 On the Web

The project's home page:

<http://slo-ist.fr/ada4autom>

The dedicated forum:

<http://slo-ist.fr/forum/index.php>

Hilscher France, author's employer:

<http://www.hilscher.fr/>

Chapter 3

About Ada

"Ada for Automation" is 100% Ada.

Hence all Ada resources are resources for "Ada for Automation".

3.1 French tutorial

You can learn a lot about Ada in French with the course of Daniel Feneuille (but you'll have to learn French):

<http://libre.adacore.com/tools/more-resources/ada-course-in-french>

For those who think that Ada is accessible only to the cream of the University or High School, the course was designed for students of the IUT of Aix.

3.2 Development studio

Without tool, man is not quite man.

If you're on Linux, there's a good chance that the tools are available in your distribution.

Look for GNAT in your Synaptic or equivalent.

Mr. Ludovic Brenta is doing an outstanding job as the maintainer Ada in Debian.

Thus, for example Synaptic enables you to easily install gnat-gps packages, gprbuild and their dependencies.

Whether you are on Linux or Windows or other AdaCore provides all the necessary tools and more:

<http://libre.adacore.com/>

3.3 Support and training

The **Ada France** association with many links:

<http://www.ada-france.org/>

Forums Ada in French :

<https://groups.google.com/forum/?fromgroups#!forum/fr.comp.lang.ada/>

<http://www.developpeur.net/forums/f227/autres-langages/autres-langages/ada/>

Forums Ada in English :

<https://groups.google.com/forum/?fromgroups#!forum/comp.lang.ada/>

AdaCore

The company **AdaCore** offers Ada support on many different hardware and software platforms.

If you plan to use "Ada for Automation" with the Hilscher cifX cards to create real-time solutions, know that the drivers for the cifX cards are available for Windows ® RTX ® or InTime (R) and for VxWorks ®, and of course Linux platforms.

Your application will therefore run in the environment of your choice.

AdaCore University

AdaCore University is e-learning site devoted to Ada and associated techniques:

<http://university.adacore.com/>

Adalog

The company **Adalog** offers too its training services for Ada.

3.4 Some references

The Wikibook "Méthodes de génie logiciel avec Ada" de Monsieur Jean-Pierre Rosen (Adalog) :

http://fr.wikibooks.org/wiki/M%C3%A9thodes_de_g%C3%A9nie_logiciel_avec_Ada

The Wikibook "Ada Programming":

http://en.wikibooks.org/wiki/Ada_Programming

The Wikibook "Ada Style Guide", a wealth of information on what to or not to do in great edifying explanations:

http://en.wikibooks.org/wiki/Ada_Style_Guide

The reference manual for Ada 2005:

http://www.adaic.org/resources/add_content/standards/05rm/html/RM-TTL.html

The "Rationale for Ada 2005" is also a good source of information:

http://www.adaic.org/resources/add_content/standards/05rat/html/Rat-TTL.html

Chapter 4

Concepts

In this chapter we try to draw a parallel between the concepts found in automation and their possible implementation with "Ada for Automation".

4.1 Security

Your application is not a so-called security application, no more nor less than if you were using a traditional controller.

Unless you use a safety PLC, which is designed for this function, safety is ensured by additional elements such as sensors and security mechanisms, such mechanical stops, or equipment such as a holding tank, a secure enclosure, etc . . .

Your application can monitor your system and report abnormal conditions by alarms or cut a main contactor, these actions must be doubled according to standardized rules.

4.2 Real Time

The concept of real time is very important in automation as in many other areas.

There are several definitions of what real time is and we will not risk to give another one.

Basically, if the control values you have elaborated in your program arrive outside a specified period of time, they are less useful . . . your truck is in the wall or soup overflowed.

Do we always need a real-time operating system? It depends on the applications. We distinguish applications that require soft real-time and those that require hard real time.

Number of applications can be satisfied with a soft real time.

If one pilot valves, pumps, regulators or autonomous drives, equipment with high time constants, the soft real-time given by general operating systems today can amply suffice.

Of course, for the axis control where fast and synchronous reaction times are required, a hard real-time operating system is necessary.

As always, engineering job is to properly analyze the problem and design a process control architecture adapted to it.

Technical constraints are rarely preponderant over those strategic or economic.

4.3 Structuration

4.3.1 Tasks

The task concept is common in automation and this fundamental notion is found in the Ada language.

Thus, such Schneider Electric PLC will have a main task, cyclic or periodic and periodic fast task.

In a Siemens PLC, we will have a cyclic main, some periodic tasks, some alarm triggered tasks, etc . . .

"Ada for Automation" implements the same type of tasks very easily since the concept is integrated into the Ada language.

Also, there is defined a cyclic or periodic main task and a periodic task, which is sufficient in many cases.

It is obviously possible to add additional tasks if needed since you have the source code.

4.3.2 Sections and organization blocks

At Schneider Electric, the tasks are running the code in sections that can be seen as functions without parameters or return values, the execution may be conditioned.

At Siemens, tasks are running the code in the organization blocks, OBs, which can also be considered as functions without parameters or return values.

In Ada, the equivalent of these sections or organization blocks is the procedure.

4.3.3 Subroutines

One can also consider the subroutines as functions without parameters or return values, so the procedures.

4.3.4 Functions

This concept is central to so-called structured programming.

With or without a return value, with or without parameters, they are found in all automation languages.

In Ada, functions without return value are procedures, with return value they are functions.

The functions can work only on their formal parameters, in this case, for a given set of parameters they provide the same return value, or work with global data.

4.3.5 Function blocks

This concept is derived from the concept of function and adding the instance data.

This instance data can save the status from call to call to the function block.

In Ada, there is the more general notion of object.

A function block is only an object with a single method.

4.3.6 Libraries

In automation, library brings together a number of elements, functions, function blocks, data types . . .

Ada offers the package, with the added notion of hierarchy.

Chapter 5

Getting started

5.1 Register on the forum!

It's always nice to find peers who discuss topics that concern us, and we strongly advise you to register on the dedicated forum: <http://slo-ist.fr/forum/index.php>

Then, answers to questions benefit all.

5.2 Acclimatization

Before you embark on this extraordinary adventure of developing your application in a high level language, you might profitably follow the course of the [AdaCore University](#). You will learn the basics of the Ada language, tool use and incidentally be working language of Shakespeare.

The Daniel Feneuille tutorial also allow you to familiarize yourself with Ada, see [French tutorial](#).

Equipped and experienced, you will then have to [obtain](#) "Ada for Automation" and libmodbus.

About libmodbus, on Linux it is normally already available in your packets, and if this is not the case, the tools are and the user manuals as well.

The following article should help you get it if you are under Microsoft Windows ®:
[A4A : compilation de libmodbus avec GNAT GPS sous Windows](#)

5.3 Get started!

Finally, choose from the sample applications provided by identifying the one that is closest to the application you want to develop. "Ada for Automation" is a framework for design automation applications in Ada. This framework is intended to allow you to create your own applications easily.

It therefore provides some examples applications, firstly possibly serve as a starting point for your and the other to illustrate the use of this application framework.

The examples are not meant to cover all of the functionality but to provide representative applications of it while remaining simple to access.

Thus, the application [App1](#) implements only the Modbus TCP protocol for communication with both the inputs and outputs of the system and as well with the monitoring tool used to demonstrate, the product PcVue ® from Arc Informatique.

This sample application 1 is a good choice for beginners. You only need your machine to learn and test.

Why PcVue ®? Because in life you have to make choices ... You are free to make another one. It is a tool that allows us to simply and quickly create a graphical interface for our applications, which is known to many of the PLC programmers, and you can get a demo version from <http://www.arcinfo.com/> [Arc Informatique].

The author also uses this demo version that has a major disadvantage, be limited to 25 input / outputs variables, which in practice is reached quickly. Also, it is necessary to juggle this limit by creating multiple graphics applications ...

We are certain that other tools can also be used. They are legion under Microsoft Windows ® and there are some Linux.

MLogic is very interesting technically, it works well but it seems stalled:

<http://mblogic.sourceforge.net/index.html>

It is written in Python / JavaScript. It should be feasible to adapt the principle to Ada Web Server.

The first version of your application is just a click away.

As it is possible to create your application on Linux as on Microsoft Windows ®, so start on the platform that you own and master. Do not cumulate difficulties.

However, we prefer of course free technologies.

Chapter 6

Guided tour

6.1 What is available today

"Ada for Automation" processes your application in a main loop that handles inputs and outputs and call user functions you write.

A library of components, a little rickety for now certainly begs to be expanded. Thank you for your contributions.

"Ada for Automation" uses other components to provide extended functionality. Reinventing the wheel, no thank you!

6.1.1 libmodbus

This library implements the Modbus protocol in its various incarnations:

- Modbus RTU in master mode and slave mode,
- and Modbus TCP in client mode and server mode.

The library is available as source code under LGPL license here:

<http://libmodbus.org/>

It is possible to use it on those platforms: Linux, Mac OS X, FreeBSD, QNX and Win32.

An Ada binding is available in "Ada for Automation" to respect to the Modbus TCP version Client and Server, as for Modbus RTU Master.

If the need to extend the binding to use Modbus RTU Slave turned out, that is, if someone requests it, this can be achieved quickly.

This binding is implemented and "Ada for Automation" and provides:

- a Modbus TCP Client which is configured by completing a simple array, this Client allows to cyclically scan a Modbus TCP servers network.
- a Modbus TCP Server which provides an interface for supervision or line PLC.
- a Modbus RTU Master configured, like the Modbus TCP Client, via a simple table, this Master allows cyclically scan a network of Modbus RTU slaves.

For more information on the Modbus protocol, including obtaining specifications, visit the dedicated website:

<http://www.modbus.org>

6.1.2 Hilscher

An Ada binding of the Hilscher cifX API is also available for the necessary functions for cyclical exchange, process image management, but also for acyclic exchanges and diagnostics.

You will need it if you would use a Hilscher cifX board that can provide the connectivity you need to manage your equipments on all major fieldbuses Real Time Ethernet or legacy.

With the cifX range available in most formats, your application may be:

- Master (or Client) on AS-Interface, CANopen, DeviceNet, PROFIBUS, EtherCAT, Ethernet/IP, Modbus TCP, PROFINET, Sercos III, or VARAN networks,
- or Slave (or Server) on CANopen, CC-Link, CompoNet, DeviceNet, PROFIBUS, EtherCAT, Ethernet/IP, Modbus TCP, POWERLINK, PROFINET, or Sercos III networks.

To be honest with you, the author works for Hilscher France. :-)

That said, nothing prevents you to provide a binding for your own communications interfaces!

6.2 Directories

So you have managed to [get](#) "Ada for Automation" and you are facing a number of directories.

app1

This is where you will find the files of the example application 1. This application implements a Modbus TCP server and some clients.

app1simu

You will find here the files for the simulation application for the example application 1.

app2

In this one you will find the files for example application 2. It implements a Hilscher PROFIBUS DP Master cifX card and a Modbus TCP server.

app3

Here you will find the files for example application 3. This application implements a Modbus TCP server and Modbus RTU Master.

book

The book "Ada for Automation" is elaborated here. This book is composed in plain text in the AsciiDoc format and processed to provide HTML or PDF files.

build

The directory for A4A construction artifacts. Having their own space, they do not contribute to congestion.

doc

The GNAT Pro Studio development environment can generate documentation from source code automatically and is set to move its production in this directory.

exe

Your executables will be placed here.

exp

Some of my experiences ... This should probably not be here.

hilscherx

The directory for the Hilscher cifX API binding. You will need it if you would use a Hilscher cifX board that can provide the connectivity you need to manage your equipment on all major fieldbuses Real Time Ethernet or legacy.

src

There beats the heart of "Ada for Automation". Feel free to dive and experiment.

test

The place to place all sorts of tests and experiments or examples.

6.3 GNAT Pro Studio Projects

GNAT Pro Studio allows to arrange the projects in a project tree with inheriting other projects. This will expand the project by adding or substituting source code files. You will please refer to GPRbuild documentation regarding the projects.

A4A/COPYING

The GPL License you should read carefully.

The following projects are common to all CLI or GUI applications.

A4A/lib_cifX32x86.gpr

The user project for the Hilscher cifX Device Driver library.
It is possible that you have to adjust it to suit your installation.
Only required if using a Hilscher cifX card.

A4A/libmodbus.gpr

The user project for the libmodbus library.
It is possible that you have to adjust it to suit your installation.

A4A/shared.gpr

An abstract project, shared by the other ones and containing common elements.

The following projects are for CLI (command line) applications.

A4A/a4a.gpr

This is the main project file.

A4A/app1.gpr

This is the application example 1 project file, it extends the main project "a4a.gpr".
Only the user program differs from the one included in the main project.

A4A/app1simu.gpr

This is the simulation application for application example 1 project file, it extends the main project "a4a.gpr".
This application replaces the task "Main_Task" of the main project by his own "Main_Task" in which was removed the management of Modbus TCP clients.

A4A/app2.gpr

This is the application example 2 project file, it extends the main project "a4a_hilscherx.gpr".
This application replaces the task "Main_Task" of the main project by his own "Main_Task".
The management of Modbus TCP clients has been replaced by management of an Hilscher cifX board.
The user program also differs from that available in the main project.

A4A/a4a_hilscherx.gpr

This one extends the main project by adding the binding to the Hilscher cifX Device Driver API and related examples.

A4A/a4a_exp.gpr

This project extends the main one with some of my experiments which may found a place in the main project. This should probably not be here.

The following figure shows the tree structure of these projects.

Dashed arrows represent relationships "with".

These relationships exist between a project that uses the services of another, those of a library as libmodbus or GtkAda for example.

The solid arrows indicate a project extension. As already mentioned upstream, it means that we add or replace files.

The "a4a.gpr" / "a4a_gui.gpr" case is a bit particular. We probably should have considered the second as an extension of the former. However, this would have further complicated the following schemes and directories to a tenuous benefit.

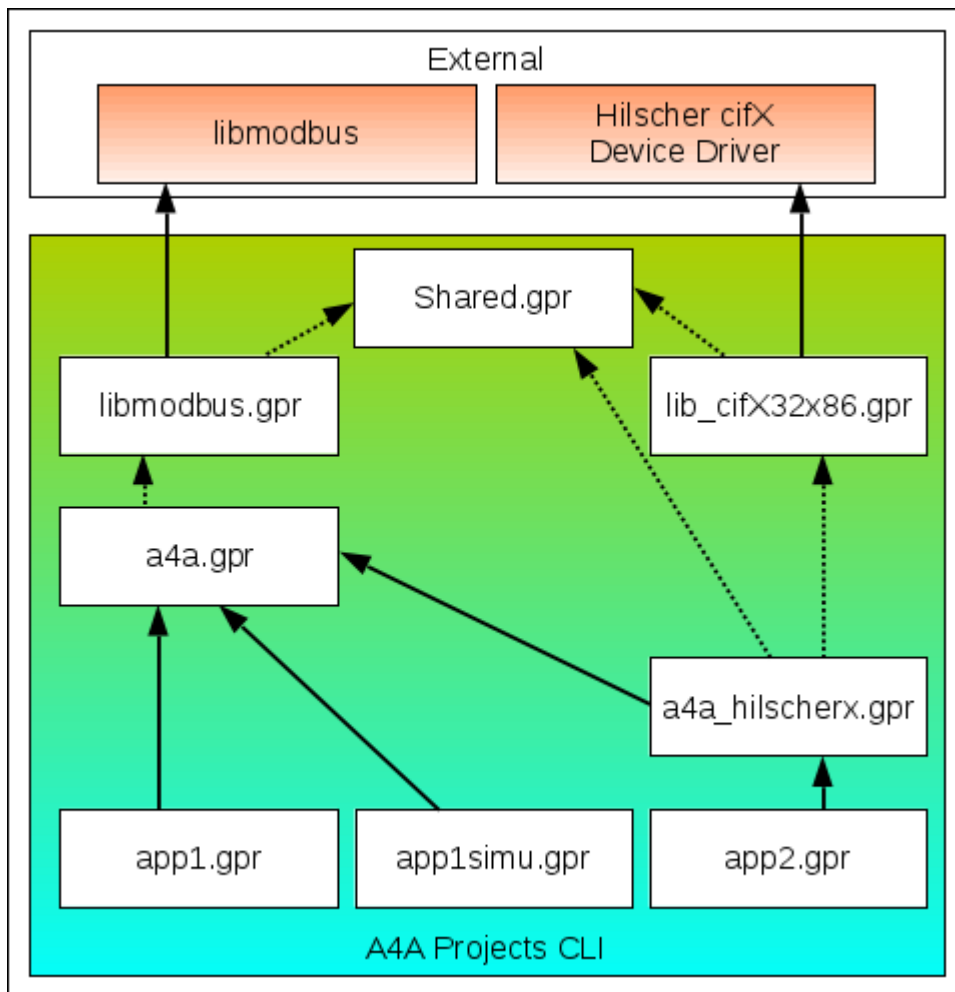


Figure 6.1: A4A CLI Projects tree

The following projects are for GUI (Graphic User Interface) applications.

A4A/a4a_gui.gpr

This is the main project file with a graphic user interface.

A4A/app1_gui.gpr

This project file of application example 1 extend main project "a4a_gui.gpr".

It is same application "app1" but with a graphical interface.

A4A/app1simu_gui.gpr

This project file of the simulation application for application example 1 extends the main project "a4a_gui.gpr".

It is same application "app1simu" but with a graphical interface.

A4A/app2_gui.gpr

It is the project file of application example 2 with GUI, it extends main project "a4a_gui_hilscherx.gpr".

A4A/a4a_gui_hilscherx.gpr

This one extends the main project by adding the GUI and the binding to the Hilscher cifX Device Driver API and related examples.

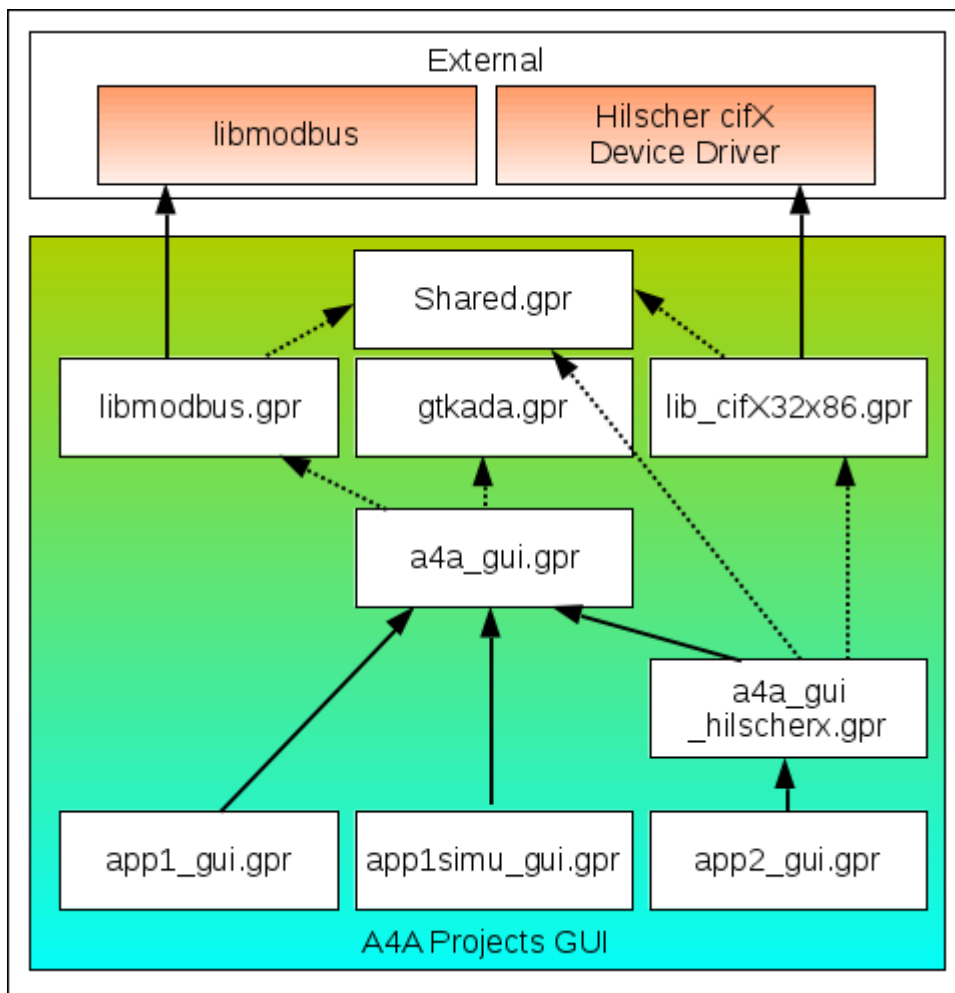


Figure 6.2: A4A GUI Projects tree

Chapter 7

Design

The purpose of this chapter is to document what is implemented, how it is and why.

7.1 General Architecture

The diagram below shows the general architecture of "Ada for Automation".

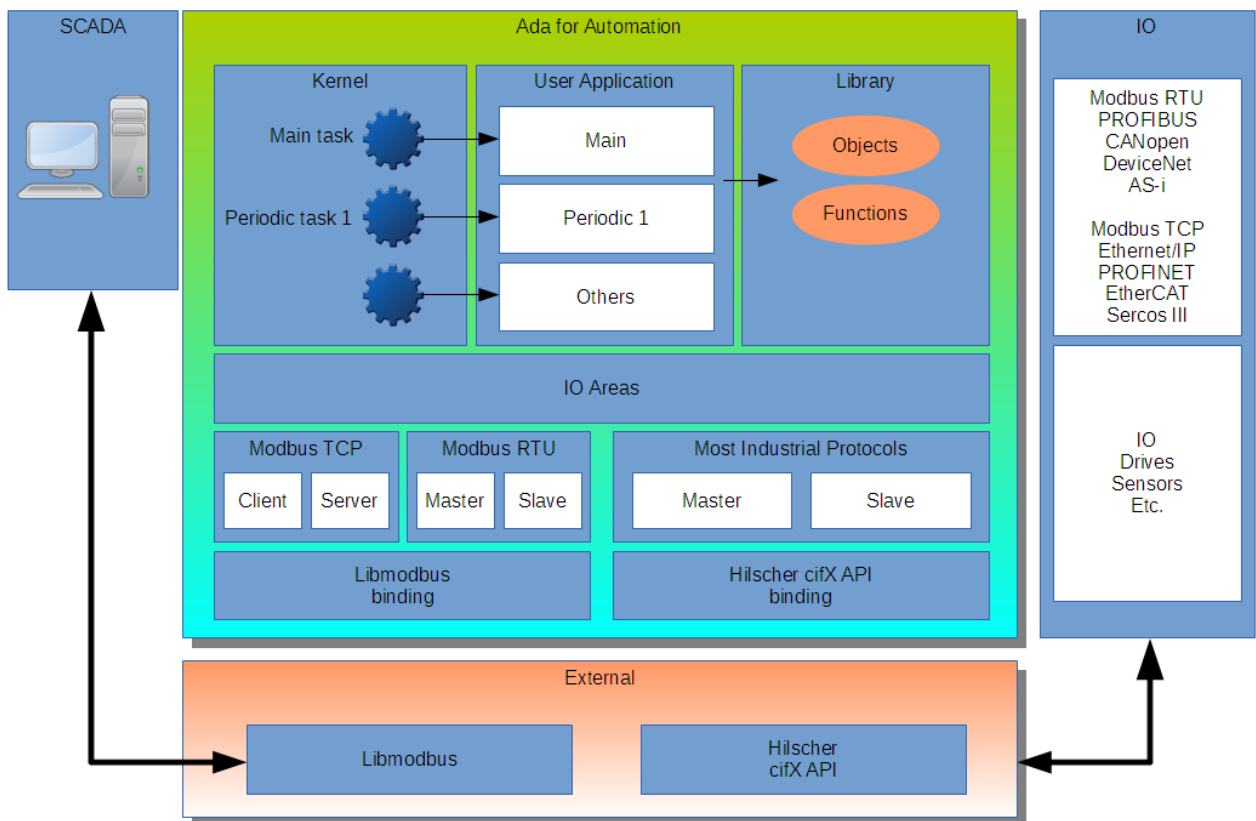


Figure 7.1: A4A General Architecture

It wants to be self-explanatory. Chapter [Design](#) is supposed to provide a detailed explanation.

A console application is defined as default

Of course, it is entirely possible to create an application with a graphical interface, for example with GtkAda, or an application with web interface with Ada Web Server (AWS).

It's so possible that it is already available for the GUI and on the "roadmap" for AWS. ;-)

7.2 Applications

7.2.1 Console application

A console application is an executable that has a command line interface.

The command line interface is a bit frustrating but has advantages:

- your application does not require any hardware (screen, keyboard, mouse) or additional libraries, it's perfect if you install the CPU that executes it in a cabinet or an electrical box,
- it is less heavy, on small configuration it counts,
- and it is more simple, which is an important advantage when you start.

This console application has its starting point in the "A4A_Console_Main" procedure. This is the main function that will create the different tasks which contribute to make the necessary actions to your control - command.

The following sequence diagram shows the main function "A4A_Console_Main" create, start and monitor the application tasks.

The main task "Main_Task" will manage others such as Modbus TCP server and clients.

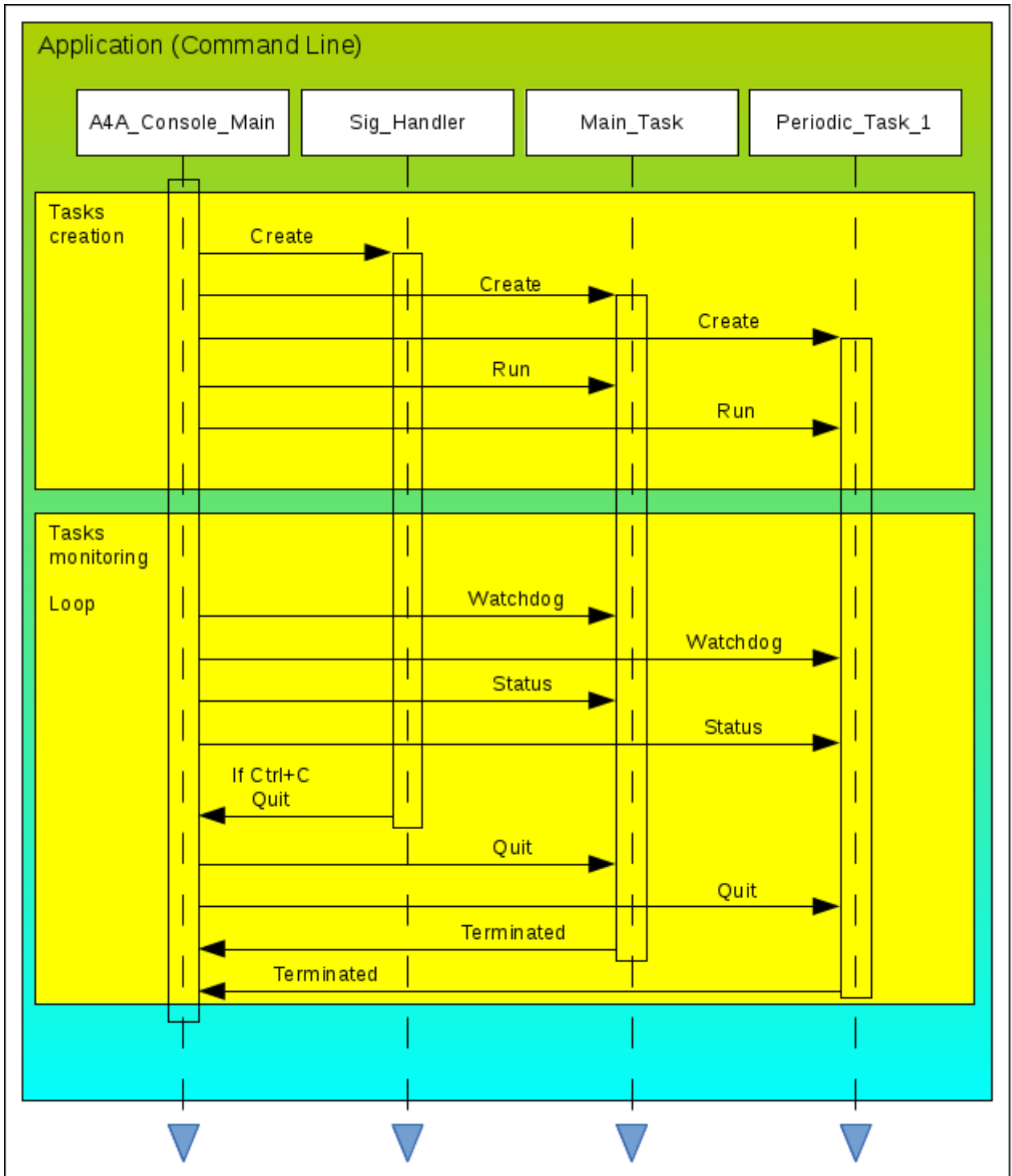


Figure 7.2: A4A Command Line Application

7.2.2 Application with Graphical User Interface

It allows a user - application interaction much richer and faster handling.

It is available both under Linux as on Windows ® because it uses the Ada binding for the GTK + library, GtkAda, this with the same Ada source.

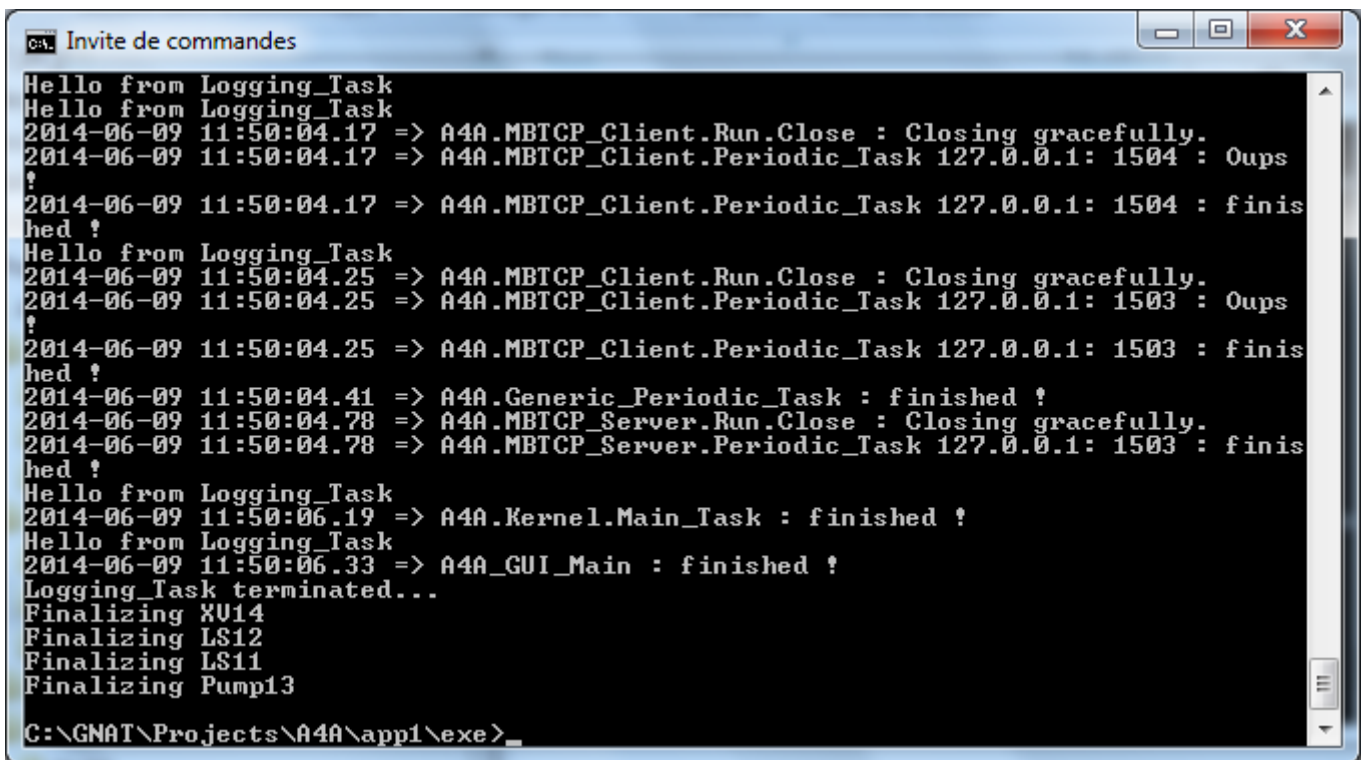
This application with HMI (or GUI) has its starting point in the procedure "A4A_GUI_Main".

The main function of the application with GUI differs from the one of the console application:

- there is no task to manage the Ctrl+C signal, task "Sig_Handler", since there is a "Quit" button,
- it starts the graphical interface which have a loop to manage keyboard and mouse events,
- it is the UI that monitors the tasks, watchdog, status visualization, start / stop ... and this periodically.

Below some screenshots with Microsoft Windows 7 ® and Debian Wheezy with the Gnome desktop environment are presented to illustrate the point.

For now, the graphics application continues to write in the window from which it is started. It is relatively verbose, it is possible of course to make it less talkative by changing the level of log messages, it is teaching by default.



```
CA Invite de commandes
Hello from Logging_Task
Hello from Logging_Task
2014-06-09 11:50:04.17 => A4A.MBTCP_Client.Run.Close : Closing gracefully.
2014-06-09 11:50:04.17 => A4A.MBTCP_Client.Periodic_Task 127.0.0.1: 1504 : Oups
?
2014-06-09 11:50:04.17 => A4A.MBTCP_Client.Periodic_Task 127.0.0.1: 1504 : finis
hed ?
Hello from Logging_Task
2014-06-09 11:50:04.25 => A4A.MBTCP_Client.Run.Close : Closing gracefully.
2014-06-09 11:50:04.25 => A4A.MBTCP_Client.Periodic_Task 127.0.0.1: 1503 : Oups
?
2014-06-09 11:50:04.25 => A4A.MBTCP_Client.Periodic_Task 127.0.0.1: 1503 : finis
hed ?
2014-06-09 11:50:04.41 => A4A.Generic_Periodic_Task : finished ?
2014-06-09 11:50:04.78 => A4A.MBTCP_Server.Run.Close : Closing gracefully.
2014-06-09 11:50:04.78 => A4A.MBTCP_Server.Periodic_Task 127.0.0.1: 1503 : finis
hed ?
Hello from Logging_Task
2014-06-09 11:50:06.19 => A4A.Kernel.Main_Task : finished ?
Hello from Logging_Task
2014-06-09 11:50:06.33 => A4A_GUI_Main : finished ?
Logging_Task terminated...
Finalizing XU14
Finalizing LS12
Finalizing LS11
Finalizing Pump13
C:\GNAT\Projects\A4A\app1\exe>
```

Figure 7.3: A4A App1 Logs in Microsoft Windows 7®

```

slos@hf-test-2: ~/Ada/A4A/app1/exe
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
2014-08-01 11:52:29.53 => A4A.MBTCP_Client.Run.Close : Closing gracefully.
2014-08-01 11:52:29.53 => A4A.MBTCP_Client.Periodic_Task 127.0.0.1: 1503 : Oups
!
2014-08-01 11:52:29.53 => A4A.MBTCP_Client.Periodic_Task 127.0.0.1: 1503 : finis
hed !
2014-08-01 11:52:29.53 => A4A.MBTCP_Client.Run.Close : Closing gracefully.
2014-08-01 11:52:29.53 => A4A.MBTCP_Client.Periodic_Task 192.168.0.100: 502 : Ou
ps !
2014-08-01 11:52:29.53 => A4A.MBTCP_Client.Periodic_Task 192.168.0.100: 502 : fi
nished !
2014-08-01 11:52:29.57 => A4A.MBTCP_Server.Run.Close : Closing gracefully.
2014-08-01 11:52:29.57 => A4A.MBTCP_Server.Periodic_Task 127.0.0.1: 1502 : Oups
!
2014-08-01 11:52:29.57 => A4A.MBTCP_Server.Periodic_Task 127.0.0.1: 1502 : finis
hed !
2014-08-01 11:52:29.93 => A4A.Generic_Periodic_Task : finished !
2014-08-01 11:52:31.53 => A4A.Kernel.Main_Task : Finished !
2014-08-01 11:52:32.08 => A4A_GUI_Main : finished !
Logging_Task terminated...
Finalizing XV14
Finalizing LS12
Finalizing LS11
Finalizing Pump13
root@hf-test-2:/home/slos/Ada/A4A/app1/exe#

```

Figure 7.4: A4A App1 Logs in Debian Wheezy

In Linux, the command "tee" is used to redirect the screen output to a file, the screen output is displayed simultaneously as recorded.

The standard graphical interface includes a button bar at the top and tabs.

In the top bar there are buttons:

- Quit: the equivalent of Ctrl + C available in the command line application that allows you to exit the application properly.
- Stop: stops processing the user program. This does not interrupt the communication tasks which thus continue to run. The outputs are set to 0, however.
- Start: starts processing the user program.

Activation of these commands display a confirmation dialog.

The action "Close Window" is equivalent to the "Quit" button as expected.

The indicator "Application status" remains desperately green point, it is not yet animated. Indeed, there is no consensus on how to display the status or what it covers as meaning.

The tabs are:

- the identity of the application,
- the general state of it,
- the status of Modbus TCP server,

- the status of Modbus TCP clients, for applications having some,
- the status of Hilscher cifX boards as appropriate.

7.2.2.1 Identity tab

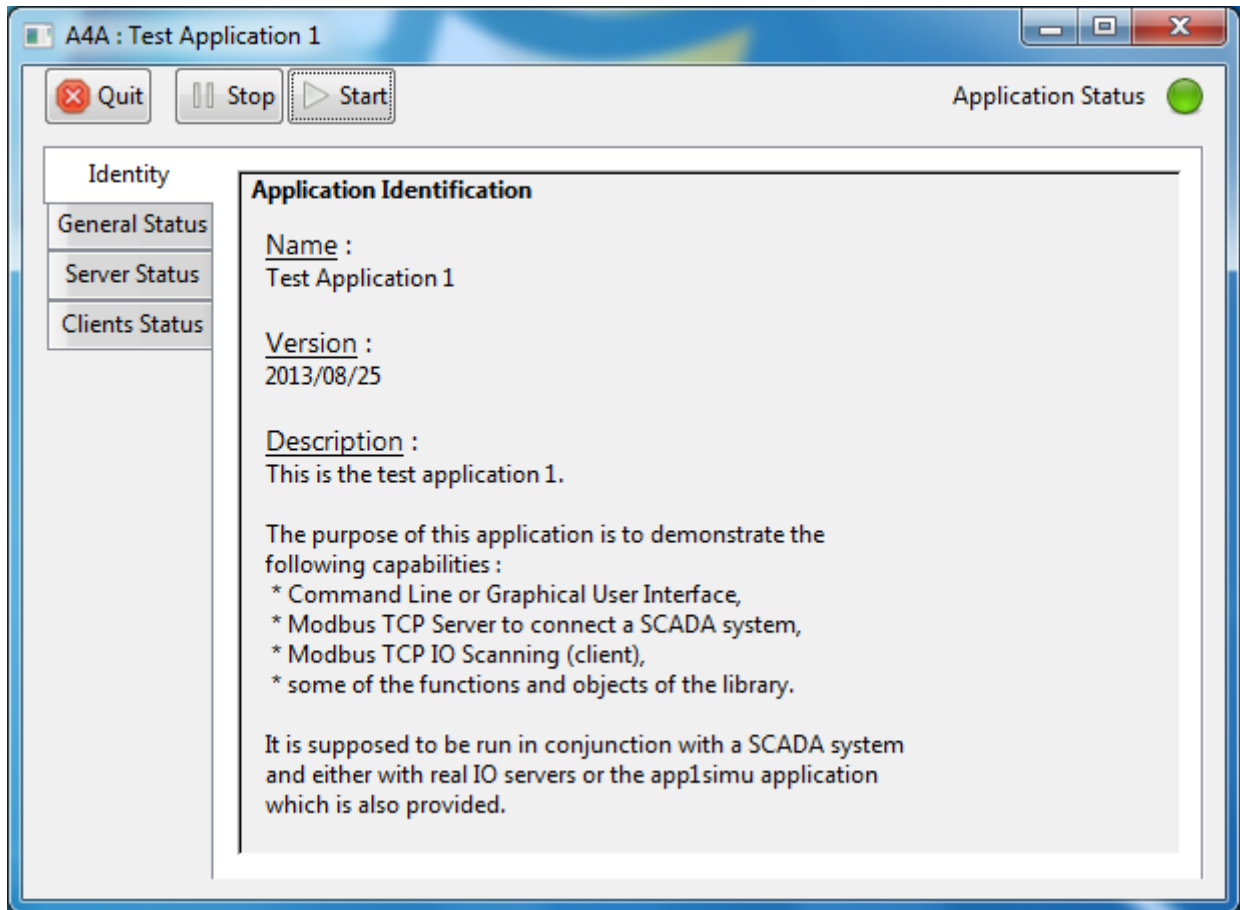


Figure 7.5: A4A App1 Identity tab in Microsoft Windows 7®

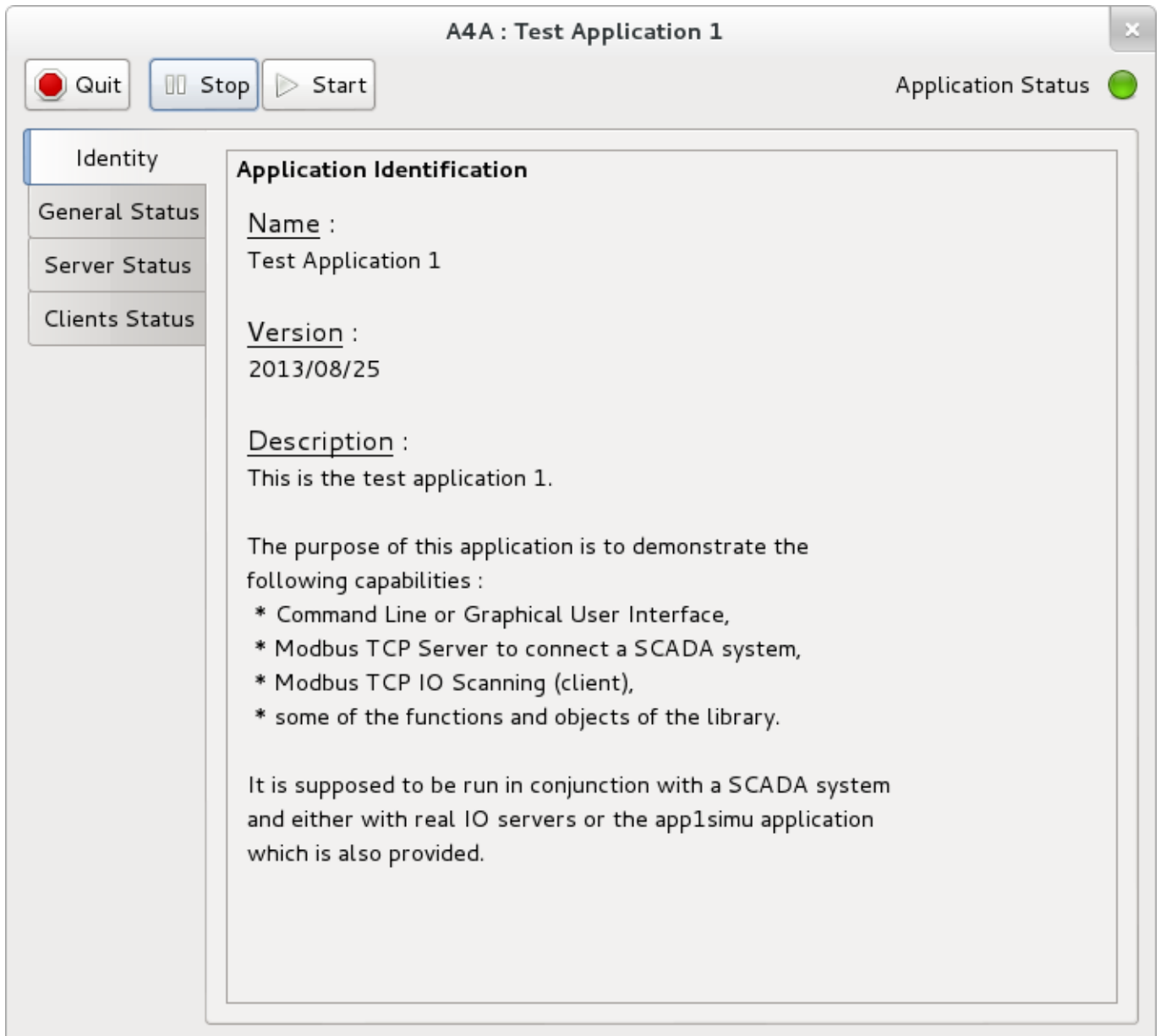


Figure 7.6: A4A App1 Identity tab in Debian Wheezy

The data presented in this tab are from the package "A4A.Application.Identification" that you can customize to your liking as of course.

Tip

It is necessary to edit private data. The interface of the package is used to obtain them.

7.2.2.2 General Status tab

A4A : Test Application 1

Quit Stop Start Application Status ●

Identity
General Status
Server Status
Clients Status

Application
Start time: 2014-07-25 16:09:56 Up time: 0h, 2m, 4s

Main Task
Task Configuration
Periodic type | Period : 50 ms

Watchdog	Running	Task duration (ms)	Scheduling Stats (delay)
●	●	Min : 0.009386000	+ 100 μs : 0
		Max : 24.795120000	+ 01 ms : 116
		Avg : 0.024424000	+ 10 ms : 1418
			+ 20 ms : 945
			+ 30 ms : 6
			+ 40 ms : 6
			+ 50 ms : 4
			+ 60 ms : 1
			+ 70 ms : 0
			+ 80 ms : 1

Periodic Task 1
Task Configuration
Period : 500 ms

Watchdog	Running	Task duration (ms)
●	●	Min : 0.000853000
		Max : 0.007254000
		Avg : 0.001771000

Figure 7.7: A4A App1 General_Status tab in Microsoft Windows 7®

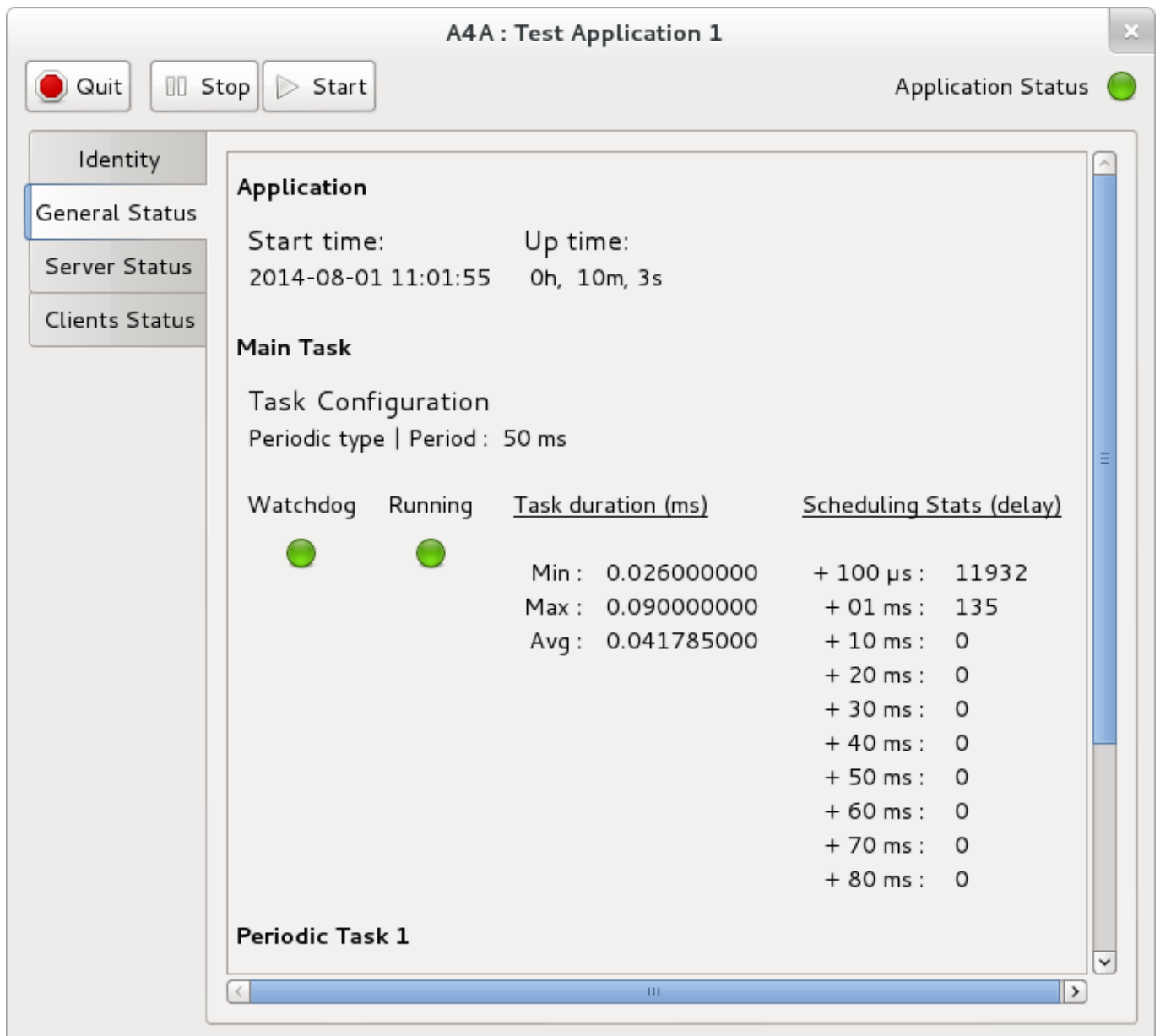


Figure 7.8: A4A App1 General_Status tab in Debian Wheezy

This tab shows:

- the application date and start time and the operating time in hours, minutes and seconds,
- for each task type, cyclic or periodic, the time or period, the state of the watchdog, the operating condition, some statistics on the execution time, min, max, average,
- to the main task, statistics about the delay in relation to the planned schedule, allowing to appreciate the real-time behavior.

Note

One must avoid any hasty judgment on these statistics.

For example, the maximum execution time is quite misleading. It is not that the job lasted longer, but rather that it was preempted by one or more other tasks.

It should be noted here that the Linux version used is with the Linux kernel patch PREEMPT_RT available in Debian Wheezy. This explains the differences in much better scheduling statistics.

However, it is possible to significantly improve the behavior of Windows ® as shown in this article:

[A4A : Améliorer le temps réel mou sous Windows](#)

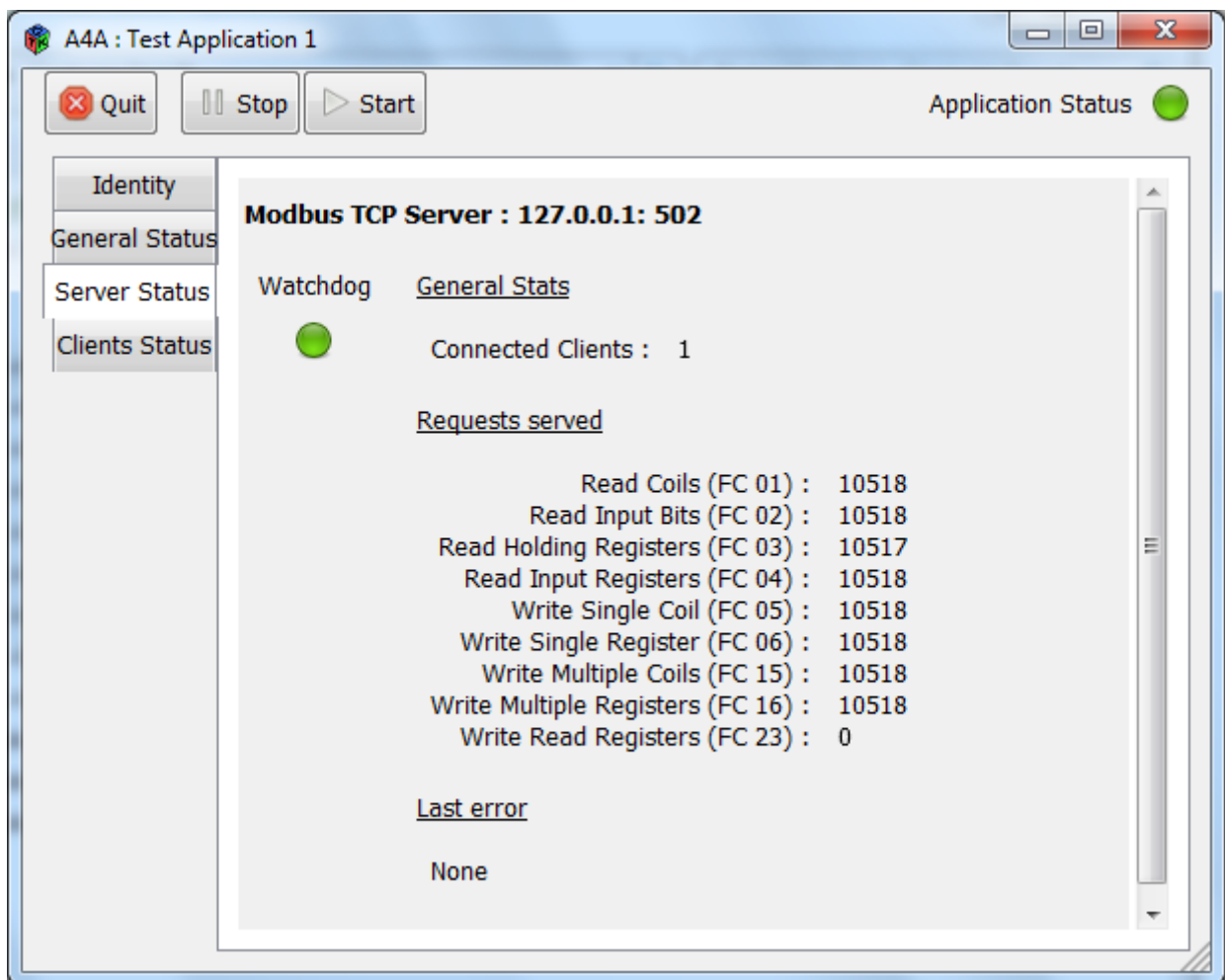
7.2.2.3 Modbus TCP Server status tab

Figure 7.9: A4A App1 MBTCPServer tab in Microsoft Windows 7®

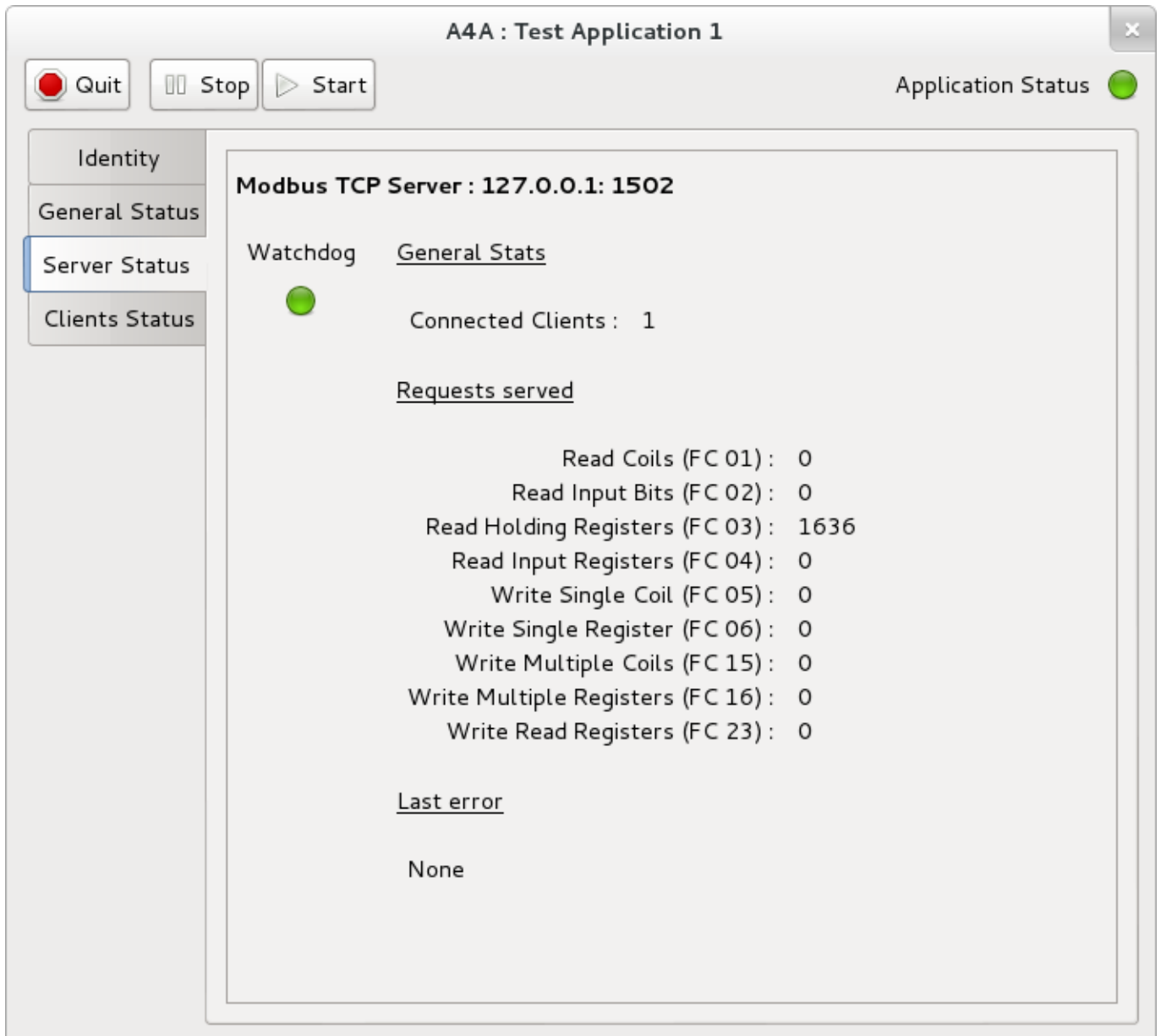


Figure 7.10: A4A Appl MBTCPSTerver tab in Debian Wheezy

This tab displays statistics of the integrated Modbus TCP server.

Its local IP address, not very helpful but it is to remain consistent with that displayed by the client, and the port are indicated.

Note

The standard Modbus TCP port is port 502. However, under Linux it is necessary to run the application with rights "root" if it uses a port below 1000. So choosing a port above 1000, by eg 1502 if we do not have to use port 502.

There is the number of connected clients and counters indicate for each of the functions supported number of requests served.

This view shows execution in Microsoft Windows 7 ®, one can very well use port 502 as shown. The connected Modbus TCP client is a Hilscher cifX board running a firmware configured as Modbus TCP client with all types of supported queries. The Hilscher client does not support function 23, read / write registers, the server "Ada for Automation" does, that can be tested with another client or the client "Ada for Automation" as shown below .

The Hilscher client is configured with a command table in which we specified all queries. Here they are executed immediately and "Ada for Automation" server is able to serve about 500 requests per second.

7.2.2.4 Modbus TCP Clients status tab

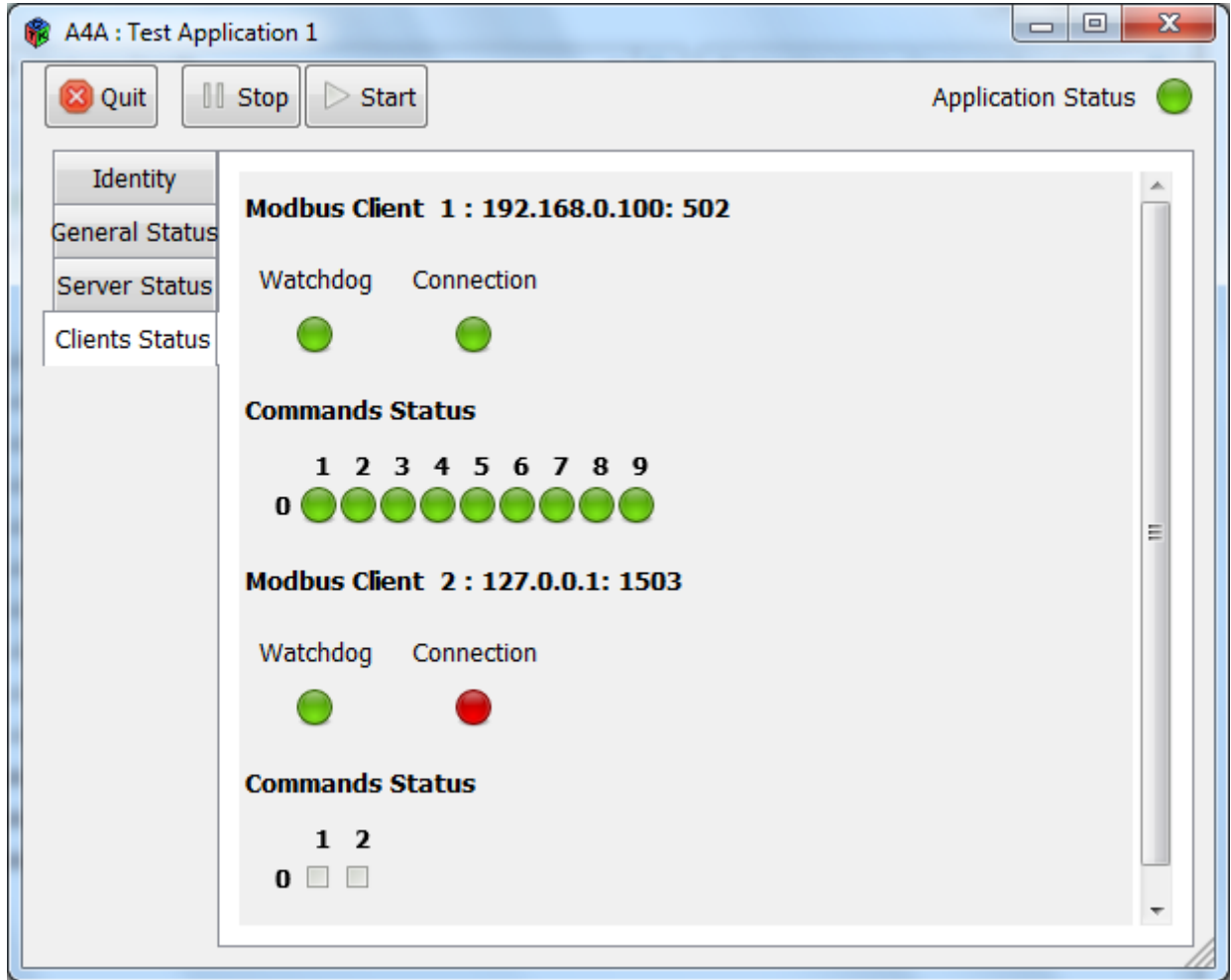


Figure 7.11: A4A App1 MBTCPClients tab in Microsoft Windows 7®

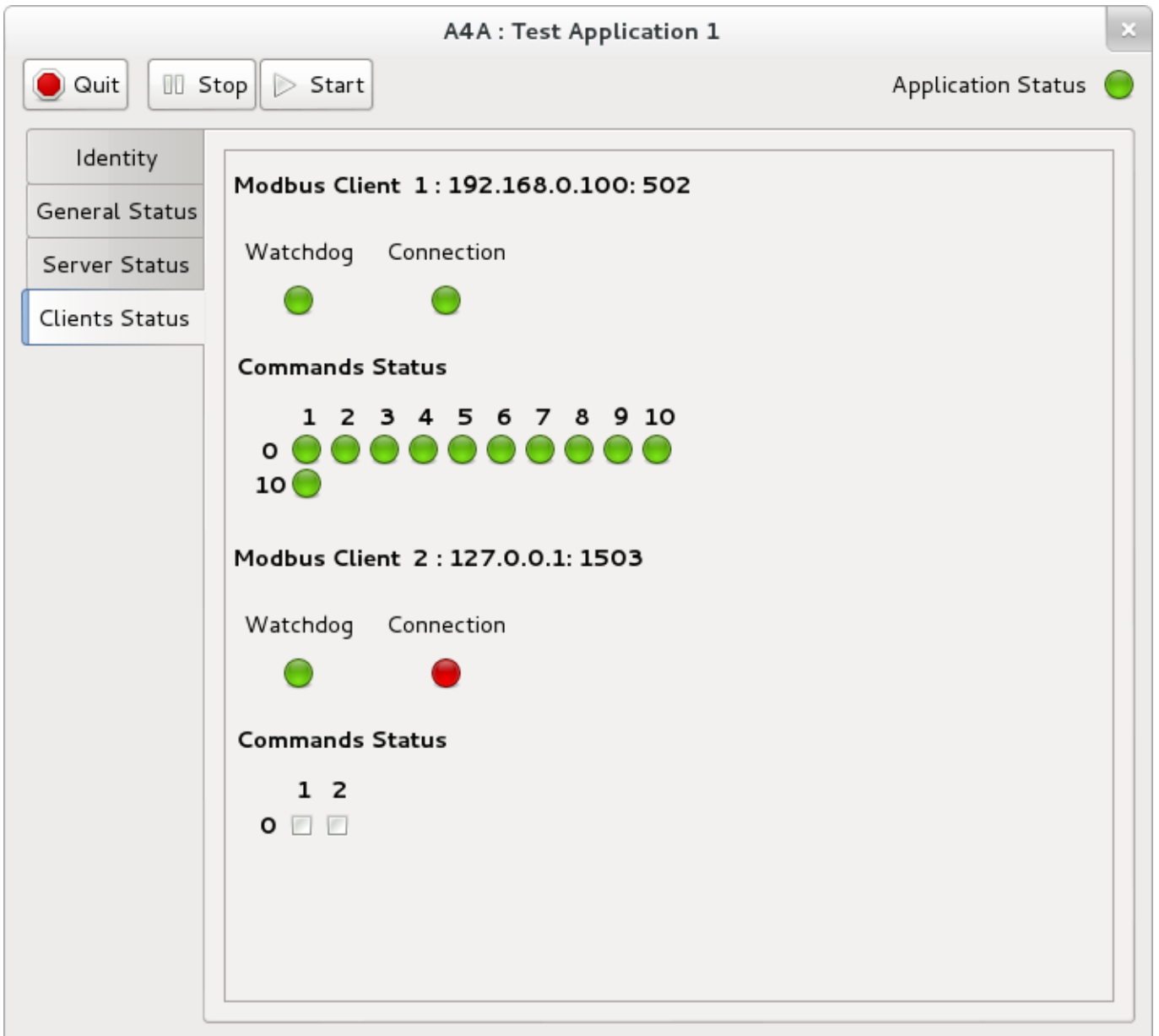


Figure 7.12: A4A App1 MBTCPClients tab in Debian Wheezy

This view shows the Modbus TCP clients at work configured as [here](#).

Note

The graphical interface automatically adjusts depending on the configuration.

7.2.2.5 Modbus RTU Master status tab

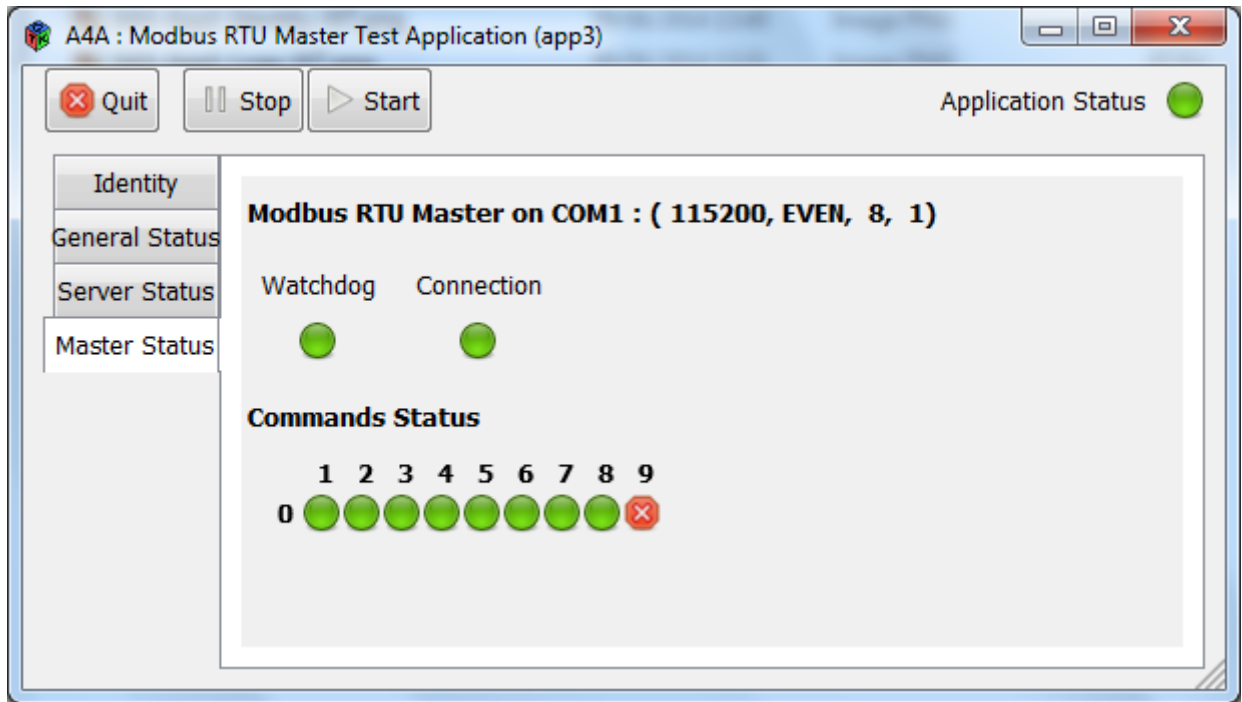


Figure 7.13: A4A App3 MBRTUMaster tab in Microsoft Windows 7®

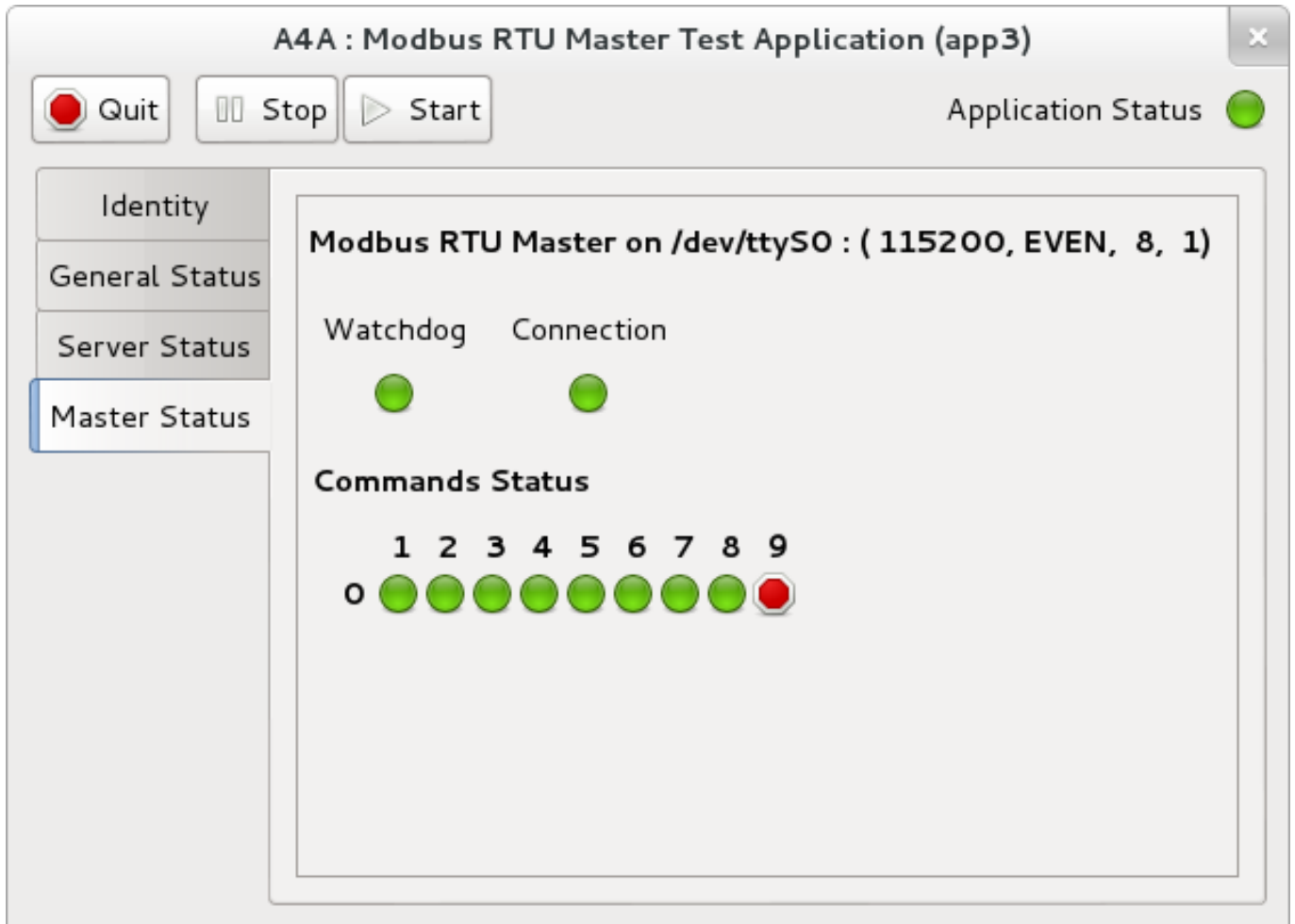


Figure 7.14: A4A App3 MBRTUMaster tab in Debian Wheezy

Here we have the Modbus RTU Master configured as [here](#), the only difference between the version for Microsoft Windows 7 ® and the Linux version is the "Device", "COM1" for one and "/ dev / ttyS0" for the other.

The "Write_Read_Registers" function is not supported by the slave used [Modbus PLC Simulator](#). Also it is disabled, which is indicated by the red cross.

Note

The graphical interface automatically adjusts depending on the configuration.

7.2.2.6 Hilscher cifX status tab

This view shows the general status of an Hilscher cifX 50-RE board running an Ethernet/IP Scanner firmware.

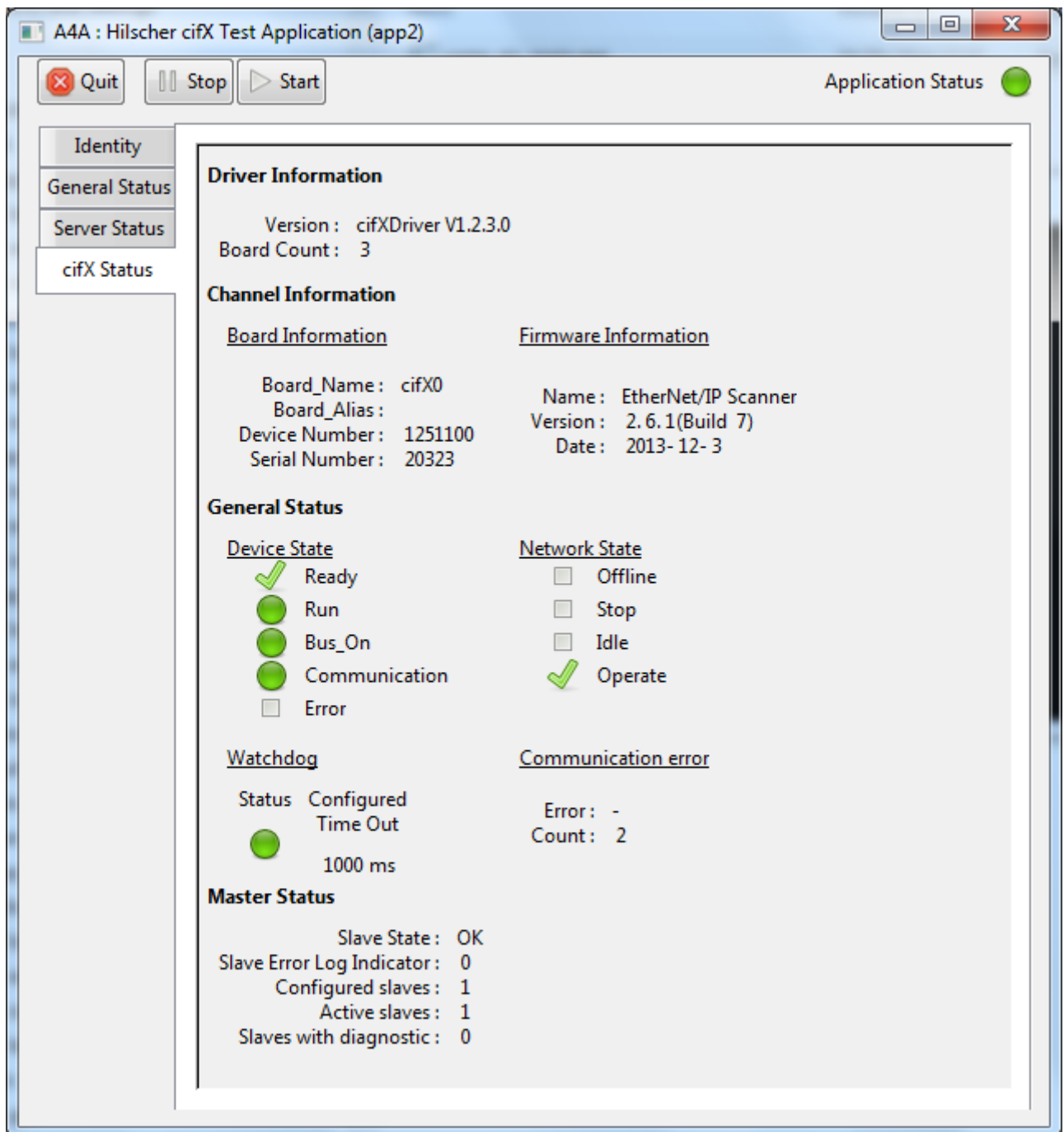


Figure 7.15: A4A App2 cifXStatus (Ethernet/IP Scanner) tab in Microsoft Windows 7®

It is also possible to use other protocols. The example application "app2" uses a Hilscher cifX as PROFIBUS DP Master. This view shows the general status of an Hilscher cifX 50-DP board running a PROFIBUS DP Master firmware.

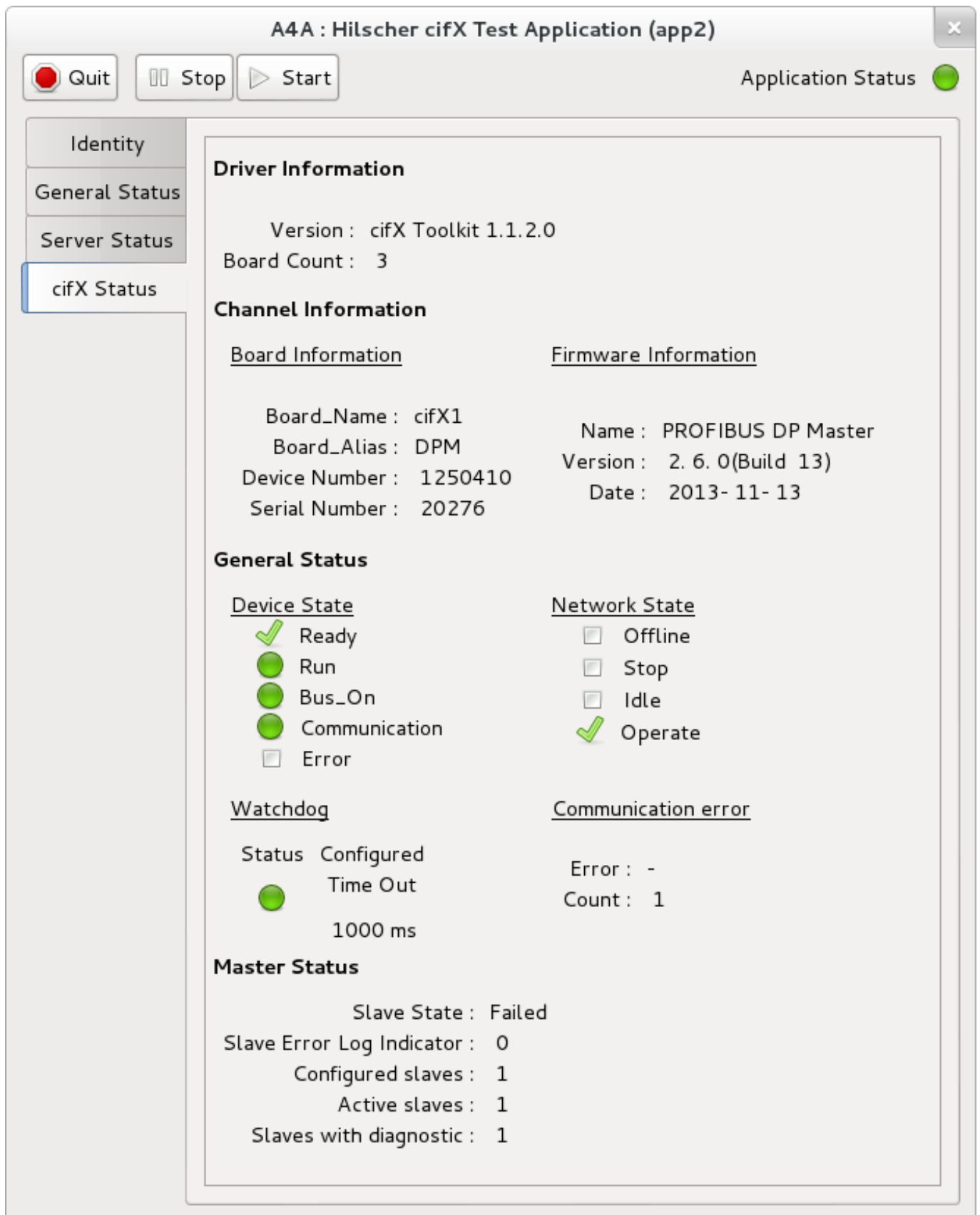


Figure 7.16: A4A App2 cifXStatus (PROFIBUS DP Master) tab in Debian Wheezy

7.3 Packages

The packages allow you to organize Ada code by grouping elements of the application, data types, data, functions and procedures, objects, tasks, other packages, etc ... in a logical hierarchy .

The children packages inherit from the parent package, things defined therein are visible to them without the need to mention a "with" clause. Thus, all children packages of "A4A" see what is defined in it and can be used without restriction.

A clause "with" defines a package dependency where it appears to the subject of the clause.

We can represent these package dependencies in a diagram, which could be done later. :-)

7.4 Tasks

We discussed in Chapter [Concepts](#) the concept of task, a task representing an instruction execution thread. Several tasks may be executed concurrently, the processor working on the instructions and the data of each task according to a schedule managed by the operating system and, in the case of Ada, by the runtime.

In Ada, it's easy to create a task, the concept is available in the language.

When designing a multi-threaded application, a recurring problem is the sharing of data and the corollary, the integrity of the shared data or consistency.

This problem is simple to solve in Ada as one can for example define a protected type that encapsulates the critical data and allows concurrent access through functions and procedures which interlock.

An example of use of a protected type is given by the scan on Modbus TCP with "Dual Port Memory", a component that allows the reading by several entities while simultaneously writing is possible only by single entity and locks the reading as writing.

In principle, the tasks in "Ada for Automation" have an interface of a protected type allowing control of the task (Quit, Run, Watchdog) and the recovery of his status.

Related tasks monitor a sign of mutual life, it is the watchdog.

So, your application is comprised of tasks, two predefined, the main task and periodic task, some ancillary tasks, and those you could add according to your needs.

7.4.1 Main_Task

The task "Main_Task" defined in package "A4A.Kernel.Main" can be configured to run cyclically or periodically.

This configuration takes place in the package "A4A.Application.Configuration".

Configured to run cyclically, it will be at the end of the iteration scheduled to run after a given period of time. For example, the task lasts 200 milliseconds. At the end of its execution, it is scheduled to run again 100 milliseconds later. Its period varies depending on the duration of the task.

If configured for periodic execution, it will be executed with a determined period, more or less precise depending on the operating system.

Periodic or cyclical? It depends on the nature of your application and the required components. If you drive only binary things, pumps, valves, lights, etc. a cyclical execution is suitable. If you use objects whose behavior depends on the concept of time as a PID controller or a signal generator, it is necessary that the computation is to the most accurate time.

This task calls the procedure "Main_Cyclic" user defined in the package "A4A.Application". This procedure is the entry point for the user application as can be the organization block OB1 at Siemens. If the status of the application allows, ie the user has activated the processing and no condition (internal or application fault) otherwise, it is in a state "on", the main user procedure is called and the outputs are controlled. Else the outputs are driven to 0.

The task "Main_Task", as defined in the basic project, runs a Modbus TCP server and Modbus TCP clients depending on the configuration to drive.

As it is possible to substitute a base project file in an extension project, it is quite possible to redefine the main task according to needs.

Thus, in the project "app1simu" as it is not necessary to implement Modbus TCP clients, the task is redefined without them.

Similarly, in the "app2" project that uses a Hilscher cifX card to communicate with PROFIBUS DP devices, the main task is different from that of the base project.

We use the term "IO Manager" to mean the component responsible for scanning I / O on the fieldbus. This can be the Modbus TCP clients, Modbus RTU master, the Hilscher card or others.

The task "Main_Task" updates the process image of the inputs, processes the application program and assigns the image of the outputs. This allows for the entire program execution duration to have a consistency of information.

The following figure shows a structure in fact very common in automation.

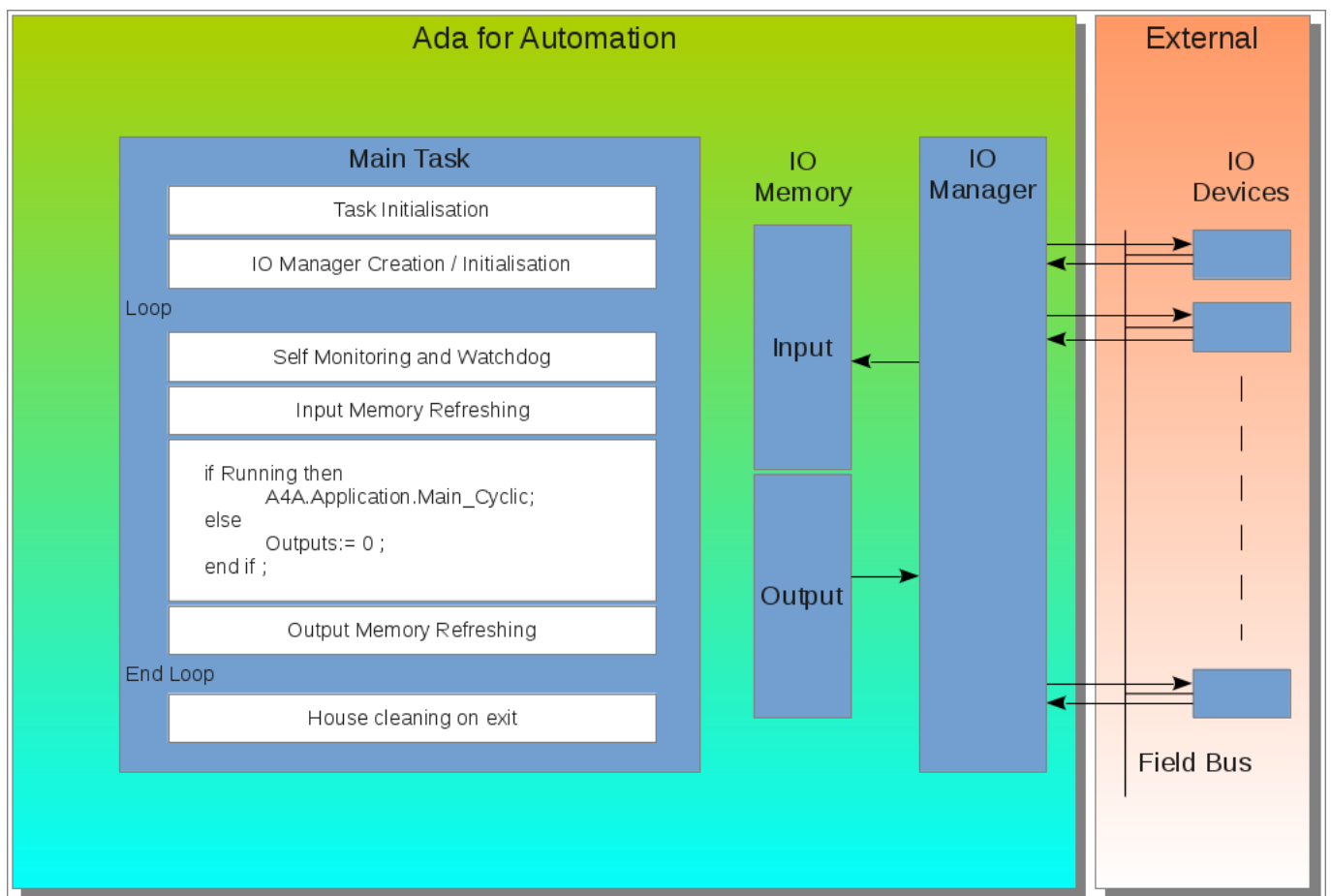


Figure 7.17: A4A Main Task

7.4.2 Periodic 1

The task "Periodic 1" can be configured to run periodically.

This configuration takes place in the package "A4A.Application.Configuration".

This task calls the procedure "Periodic_Run_1" user defined in the package "A4A.Application".

Today, the data (DPM, user objects ...) are shared between "Main_Task" and "Periodic 1" tasks.

**Warning**

It would be better to handle this via a protected object.

7.4.3 Sig_Handler

This appendix task is started by the console application.

Its only job is to wait for the interrupt triggered by the user who wants to quit the application cleanly with Ctrl + C.

7.4.4 Clock_Handler

This appendix task is started by the task "Main_Task".

In automation, timers are often used. If each timer instance called the "Ada.Real_Time.Clock" function in a quite big user program, the author fears that this is not desirable.

This is a task that runs periodically, with a period which therefore defines the refreshing of the clock data. It is this data that is used by the timers.

7.5 Modbus TCP IO Scanning

For each Modbus TCP server, one client task is used which will be created automatically by the "Main_Task" task from the supplied configuration. Thus, if a server is slow or unresponsive, scanning of others is always assured optimally.

Each client task runs periodically, the period being defined by the parameter "Task_Period_MS" in the configuration of the task.

Each command has a multiplier, the "Period_Multiple" parameter, which thus defines the period of this command ("Task_Period_MS" x "Period_Multiple") and an offset, defined by the "Shift" parameter, which allows differentiating in time several orders maturing so that the server is not saturated.

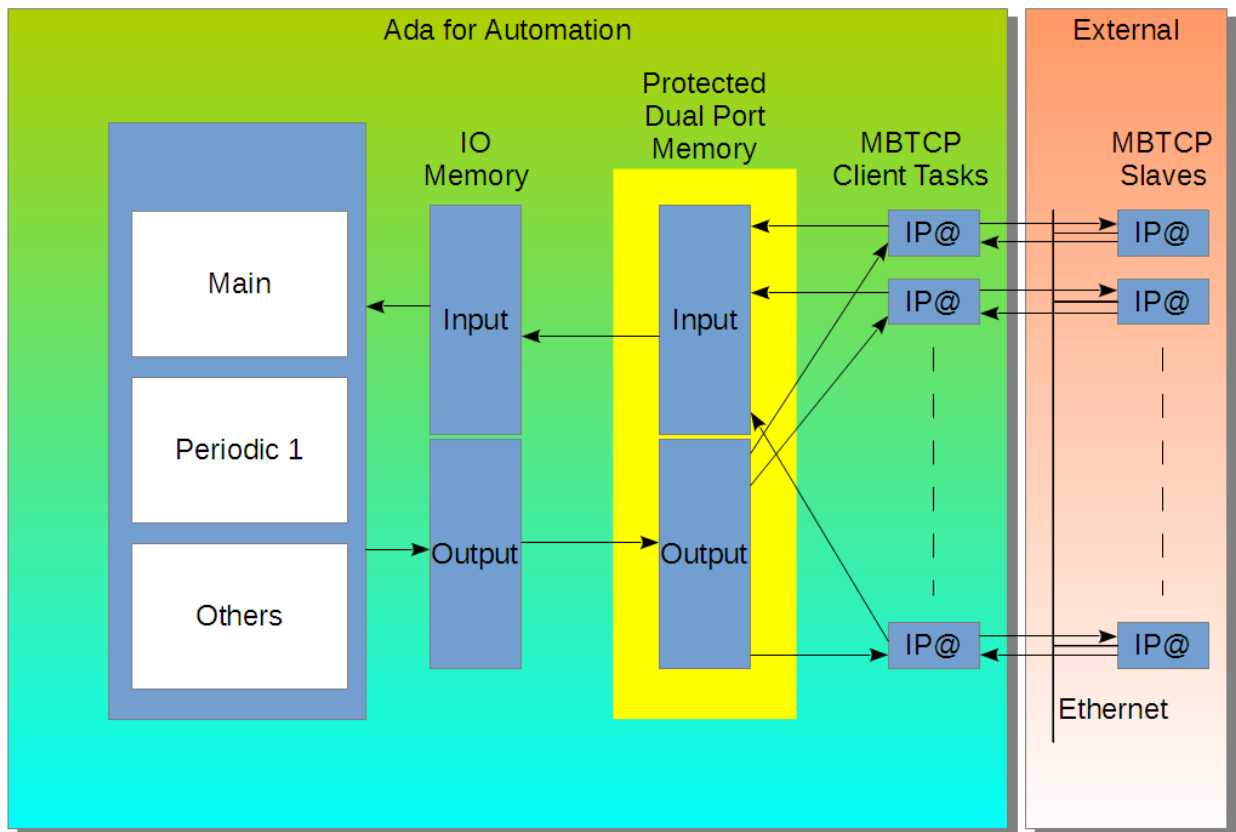


Figure 7.18: A4A Modbus TCP Clients Architecture

The "Dual Port Memory" is actually a protected object that encapsulates an array of registers for input and one for output. The task "Main_Task" manages the update of memory areas Input / Output from the DPM, I / O areas to which the user program has access.

The data types used in the Modbus protocol are registers (or 16-bits word) and booleans (coils and status bits). The protocol defines two data models, wherein the one-bits data and registers data are disjoint, and the other where there is overlap. We prefer the first model and has been defined areas for registers and areas for booleans, respectively registers DPM and booleans DPM.

The diagram is so incomplete as there is an identical construction for booleans not shown.

In terms of configuration, it is performed in package "A4A.Application.MBTCP_Clients_Config" like hereafter for the example application "App1":

```
with A4A.MBTCP_Client; use A4A.MBTCP_Client;
with A4A.Protocols; use A4A.Protocols.IP_Address_Strings;
with A4A.Protocols.LibModbus; use A4A.Protocols.LibModbus;

package A4A.Application.MBTCP_Clients_Config is

-----
-- Modbus TCP Clients configuration
-----

-- For each Modbus TCP Server define one client configuration task
```



```

Config1 : aliased Client_Configuration :=
  (Command_Number   => 9,           -- ❶
   Enabled          => True,        -- ❷
   Debug_On        => False,       -- ❸
   Task_Period_MS  => 10,          -- ❹
   Retries         => 3,           -- ❺
   Timeout         => 0.2,         -- ❻

   Server_IP_Address => To_Bounded_String("192.168.0.100"), -- ❼
   -- 127.0.0.1 / 192.168.0.100

   Server_TCP_Port  => 502,        -- ❽
   -- 502 Standard / 1502 PLC Simu / 1504 App1Simu

```

- ❶ **Command_Number** specifies the number of commands executed on the server and sizes table **Commands**.
- ❷ **Enabled** enables or disables the scan on the server.
- ❸ **Debug_On** enables or disables debugging mode on libmodbus, which activates or not the trace of communication.
- ❹ **Task_Period_MS** specifies the own period of the client task.
- ❺ **Retries** specifies the number of retries before declaring the command faulty.
- ❻ **Timeout** specifies the maximum response time.
- ❼ **Server_IP_Address** indicates the address of the server to which the client should connect.
- ❽ **Server_TCP_Port** indicates the port on which the server listens for requests, 502 is the standard port.

```

Commands =>
  (
    --
    --
    -- Action Enabled Multiple Shift Number Remote Local
    1 =>
      (Read_Input_Registers, True, 10, 0, 10, 0, 0),
    2 =>
      ( Read_Registers, True, 20, 0, 10, 0, 10),
    3 =>
      ( Write_Registers, True, 30, 0, 20, 20, 0),
    4 =>
      ( Read_Bits, True, 30, 1, 16, 0, 0),
    5 =>
      ( Read_Input_Bits, True, 30, 2, 16, 0, 32),
    6 =>
      ( Write_Register, True, 30, 3, 1, 50, 30),
    7 =>
      ( Write_Bit, True, 30, 4, 1, 0, 0),
    8 =>
      ( Write_Bits, True, 30, 5, 15, 1, 1),

    9 => (Action           => Write_Read_Registers, -- ❶
         Enabled         => True,
         Period_Multiple => 10,
         Shift           => 5,
         Write_Number    => 10,
         Write_Offset_Remote => 40,
         Write_Offset_Local  => 20,
         Read_Number     => 10,
         Read_Offset_Remote  => 10,
         Read_Offset_Local  => 20)
  )
);

```

```

Config2 : aliased Client_Configuration :=                -- ❷
  (Command_Number    => 2,
   Enabled           => False,
   Debug_On         => False,
   Task_Period_MS   => 100,
   Retries          => 3,
   Timeout          => 0.2,

   Server_IP_Address => To_Bounded_String("127.0.0.1"),
   Server_TCP_Port   => 1503, -- My own MBTCP server

   Commands =>
     (
      --
      --
      -- Action Enabled Multiple Shift Number Remote Local
      1 =>
        ( Read_Registers, True, 10, 0, 10, 0, 0),
      2 =>
        ( Write_Registers, True, 30, 1, 10, 0, 0)
     )
  );

-- Declare all clients configuration in the array
-- The kernel will create those clients accordingly

MBTCP_Clients_Configuration : Client_Configuration_Access_Array :=
  (1 => Config1'Access, -- ❸
   2 => Config2'Access);

end A4A.Application.MBTCP_Clients_Config;

```

- ❶ For command "Write_Read_Registers" we note that it requires some extra parameters compared to others. This shows both that the structure of the command parameters is determined by the identifier of the command, and secondly that it is possible to name the fields of the structure.
- ❷ This configuration is defined only as an example.
- ❸ Finally, we have to fill the table of client configurations.

7.6 Modbus TCP Server

Modbus specification defines a data model that contains four tables consisting of bits for inputs (Discrete Inputs) read-only and outputs / flags (Coils) read / write, or registers, 16-bit words, for analog inputs (Input registers) read only and the internal registers (Holding registers) read / write.

Each table has 65536 items.

There are two possibilities, one is that the four tables are disjoint, the other where they are overlaying. The implementation in "Ada for Automation" is the first type.

Table 7.1: Data Model

Table	Type	Access	Description
Input Bits	Bit	R	Discrete Inputs
Coils	Bit	R / W	Coils
Input Registers	Register	R	Input Registers or analog inputs
Holding Registers	Register	R / W	Holding Registers or analog outputs

The Modbus TCP server is managed by a dedicated task that responds to client requests regardless of tasks dealing with automation.

A locking mechanism is used to manage data consistency blocking readings during the execution of a write operation.

The architecture looks more or less to the one organized for clients. There we find the DPM and the corresponding IO memory areas.

The task "Main_Task" manages the update of memory areas Input / Output from the DPM, but also copies the I / O data in the data server to allow visualization of the raw data.

7.7 Modbus RTU IO Scanning

The design is substantially identical to that of "Modbus TCP IO Scanning".

Configuration is the same and differs only in the parameters of the serial port used by the master and the bus address of the slave in the command table.

On a Modbus bus there can be only one master. The master cyclically polls the slaves as configured in the table.

Instead of having a client task by server as in the case Modbus TCP, one single task performs the function master.

Configuration is performed in package "A4A.Application.MBRTU_Master_Config" like this for the example application "App3":

```
with A4A.MBRTU_Master; use A4A.MBRTU_Master;
with A4A.Protocols; use A4A.Protocols.Device_Strings;
with A4A.Protocols.LibModbus; use A4A.Protocols.LibModbus;

package A4A.Application.MBRTU_Master_Config is

-----
-- Modbus RTU Master configuration
-----

Config : aliased Master_Configuration :=
  (Command_Number => 9,

   Enabled         => True,
   Debug_On       => False,
   Task_Period_MS  => 10,
   Retries        => 3,
   Timeout        => 0.2,

   Device          => To_Bounded_String("/dev/ttyS0"),
   -- "COM1" on Windows
   -- "/dev/ttyS0" on Linux

   Baud_Rate      => BR_115200,
   Parity         => Even,
   Data_Bits      => 8,
   Stop_Bits      => 1,

   Commands =>
     (
       --
       --
       -- Action Enabled Multiple Shift Slave Number Remote Local
       1 =>
         (Read_Input_Registers, True, 10, 0, 2, 10, 0, 0),
       2 =>
         ( Read_Registers, True, 20, 0, 2, 10, 0, 10),
```

```

3 =>
  (   Write_Registers,   True,      30,    0,    2,    20,    20,    0),
4 =>
  (   Read_Bits,        True,      30,    1,    2,    16,    0,    0),
5 =>
  (   Read_Input_Bits,  True,      30,    2,    2,    16,    0,    32),
6 =>
  (   Write_Register,   True,      30,    3,    2,    1,    50,    30),
7 =>
  (   Write_Bit,        True,      30,    4,    2,    1,    0,    0),
8 =>
  (   Write_Bits,       True,      30,    5,    2,    15,    1,    1),

9 => (Action           => Write_Read_Registers,
     Enabled          => False,
     Period_Multiple => 10,
     Shift            => 5,
     Slave            => 2,
     Write_Number     => 10,
     Write_Offset_Remote => 40,
     Write_Offset_Local  => 20,
     Read_Number      => 10,
     Read_Offset_Remote => 10,
     Read_Offset_Local  => 20)
)
);

end A4A.Application.MBRTU_Master_Config;

```

This is similar to the Modbus TCP client configuration except the serial port settings and the slave address added in commands.

7.8 Data Flow

The following diagram shows the flow of data in the application example 1, detailed in Chapter [devoted](#).

The memory areas are defined in package "A4A.Memory." The "Dual Port Memory" is automatically allocated in package "A4A.Kernel" according to the necessary size.

The black flows at the initiative of Modbus TCP clients.

Flows in green correspond to the scan of inputs and outputs on Modbus TCP servers.

In red are represented the flows managed by the task "Main_Task" of package "A4A.Kernel."

Finally, the blue flows are organized by the user in the functions of the package "A4A.User_Functions" "Map_Inputs, Map_HMI_Inputs, Map_Outputs, Map_HMI_Outputs" called in the main procedure "Main_Cyclic", itself called by the task "Main_Task".

In the case of the application example 2, "app2" aptly named, just replace in this scheme Modbus TCP clients by Hilscher cifX board.

The DPM of the board also includes a process image with an input area and an output area.

In the scheme of application example 3, "app3", simply replace the Modbus TCP clients by master Modbus RTU.

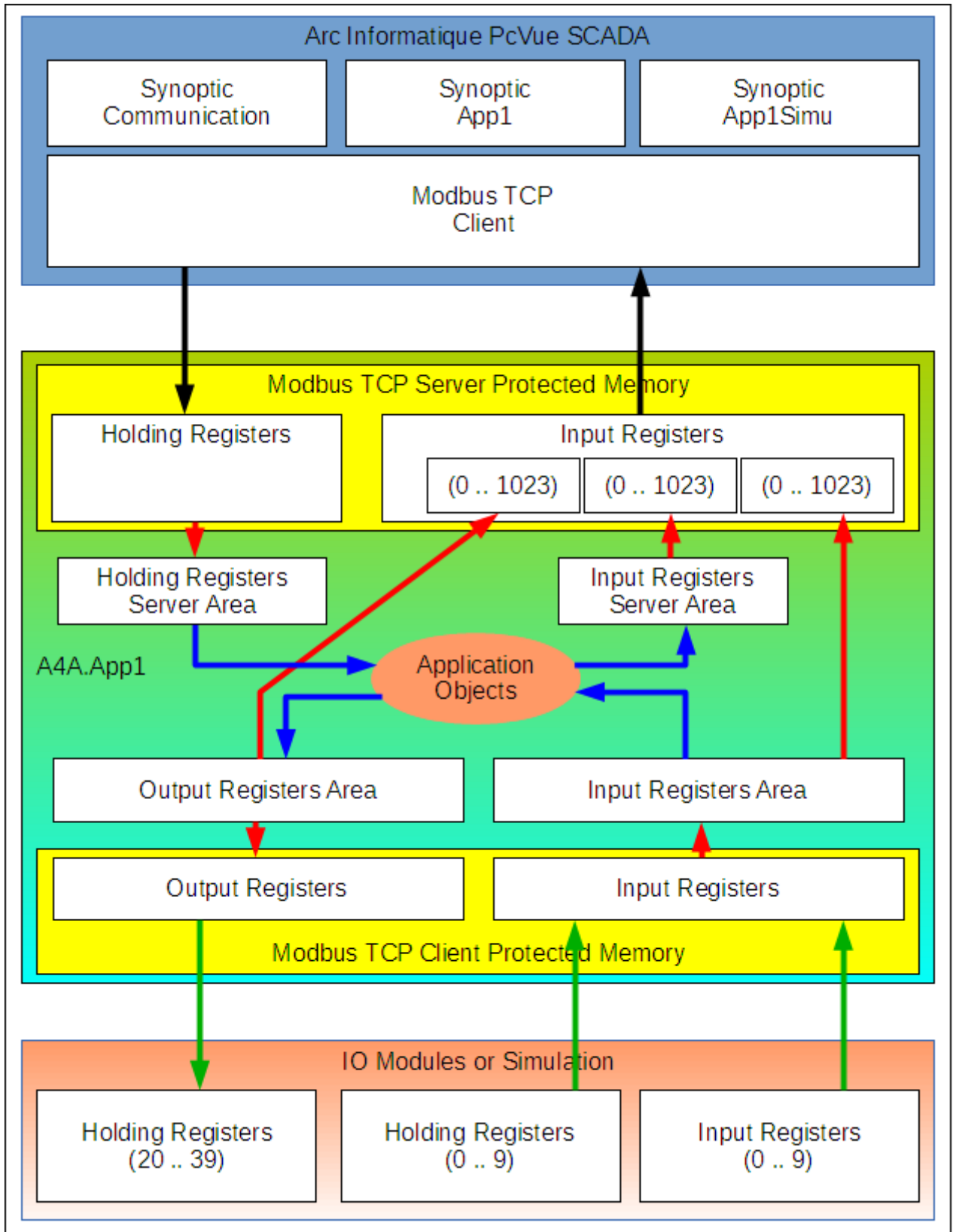


Figure 7.19: A4A Data flow with a nominal application

7.9 Interface with the user program

This interface defines the interactions between the framework and the user program.

The framework calls the functions of the user program defined in package "A4A.Application" as already seen in the paragraphs on tasks "Main_Task" and "Periodic 1".

We have already seen the procedures "Main_Cyclic" and "Periodic_Run_1."

The procedure "Cold_Start" is also called by the task "Main_Task" before entering the cyclical execution loop when initializing the task is completed, so that the user program can make itself some initialization.

An example of use is provided by the "App2" application.

The procedure "Closing" is itself called by the task "Main_Task" when the application is terminated, by the signal "Ctrl + C" in the case of the console application or the "Quit" button in the GUI application before the task closes the communication means.

The "App2" application again provides an example.

User programs can generate an exception such as division by 0. This exception is normally managed and a flag "Program_Fault" then stood up.

The "Program_Fault" function allows the framework to determine the status of this flag and this has the same effect as the "Stop" button.

Except for the fact that there is no other way to restart than to restart the application.

Chapter 8

Application Example 1

The "App1" implements only the Modbus TCP protocol for communication with both the inputs and outputs of the system and the SCADA system, the product PcVue ® from Arc Informatique, used for the demonstration.

The application runs on both Linux and Windows ® thanks to libmodbus and Ada, however, for simplicity we will stick to the majority platform (for now), especially since the used SCADA, for man - machine interface, runs on Windows ®.

An article, in French, also presents the application and it shows PcVue in action
<http://slo-ist.fr/ada4automation/a4a-exemple-dapplication-app1>

The example application 1 is provided in the "app1" folder.

It therefore shows how to implement a basic application with:

- Modbus TCP clients,
- a Modbus TCP server to connect a SCADA system for example,
- the implementation of conventional automation functions.

In terms of the operative part, we do not have. Also, there are three options:

- you have the operative part and have some money to buy electrical equipment, I/O modules, sensors and actuators ...
- you download a PLC simulator as [Modbus PLC Simulator](#),
- you use the operating part simulation application developed concurrently!

For our part, we prefer the simulation application and that is why we have developed it as well.

The simulation application to application example 1 is provided in the "app1simu" folder.

8.1 Functional specifications

Here below the functional specifications of this application.

8.1.1 Object

This application relates to the automatic management of watering the garden of my stepfather.

8.1.2 Description

A tank stores the rainwater collected from the roof of the house, in the order of 700 to 900 liters per square meter per year in the Loire.

The tank can be supplied by the municipal water supply to compensate for a lack of rainfall, a normally closed solenoid valve controls the flow.

Garden is served by a hose system and a feed pump.

The tank is equipped with:

- a level sensor providing a digitized analog measurement,
- a digital level sensor above, the information is used to close the water supply valve, not to fill the pond below,
- a digital level sensor down, the information is used to stop the pump to protect it from dry run.

The analog level information is processed to provide the automation thresholds with hysteresis, the thresholds are used for the management of actuators:

- XHH : closes the solenoid valve,
- XH :
- XL : opens the solenoid valve,
- XLL : stops the pump.

The information of the digital level sensor (SL and SH) are arranged on the one hand in the actuator control chain (safety function) and, on the other hand, are feeded to the control for the generation of the corresponding alarms.

We have : $SL < XLL$ and $SH > XHH$

The valve is instrumented, two limit switches, open and closed, are controlled.

The pump has a contactor feedback.

8.1.3 Operating Modes

The facility has two operating modes, manual mode and automatic mode.

The selection of the operating mode is done on the control panel via a switch.

The manual mode allows testing of the installation or filling the tank early.

8.1.3.1 Manual Mode

The valve can be controlled by means of commands available on the HMI. The XHH threshold closes the valve.

The pump can also be controlled via the commands available on the HMI. The XLL threshold stops the pump.

8.1.3.2 Automatic Mode

Watering is started on a schedule and depending on humidity sensors. (in a later version because we do not have enough variables at the HMI level since it is a demo version . . .)

When the watering is operating, the XL threshold triggers the opening of the valve, the threshold XHH closing of it.

The pump is switched on during watering as long as the XLL threshold is covered.

8.1.4 Human - Machine Interface

The HMI displays data from the control, status of inputs and outputs, alarms, trend curve on the level and allows the control of the actuators in manual mode.

It also allows the setting of the watering schedule.

8.1.5 Control panel

Found on the control panel a Auto / Manual switch and a push button to acknowledge faults.

8.1.6 Instrumentation

- a Level Transmitter or LT (Level Transmitter, LT10)
- a Level Switch or LS (Level Switch, LS11) with a threshold high SH,
- a Level Switch or LS (Level Switch, LS12) with a threshold low SL.

8.1.7 Inputs / Outputs

8.1.7.1 Analog Inputs

- Level Transmitter LT10

8.1.7.2 Digital Inputs

- LS11_SH
- LS12_SL
- Pump13_FB, pump contactor feedback
- V14_ZO, valve limit switch open
- V14_ZF, valve limit switch closed
- Auto, switch Auto / Manu
- Manu, switch Auto / Manu
- Ack_Fault, Fault Ack button

8.1.7.3 Digital Outputs

- P13_Coil, pump contactor coil
- V14_Coil, solenoid valve coil

8.1.8 Human - Machine Interface

8.1.8.1 HMI ⇒ Control

Valve and pump manual commands.

8.1.8.2 Control ⇒ HMI

The status of the sensors and actuators.

8.1.9 Simulation

Failing operative part, a simulation application is developed for testing, acceptance tests and user training. :-)

8.2 Overview

Here is a diagram showing the architecture:

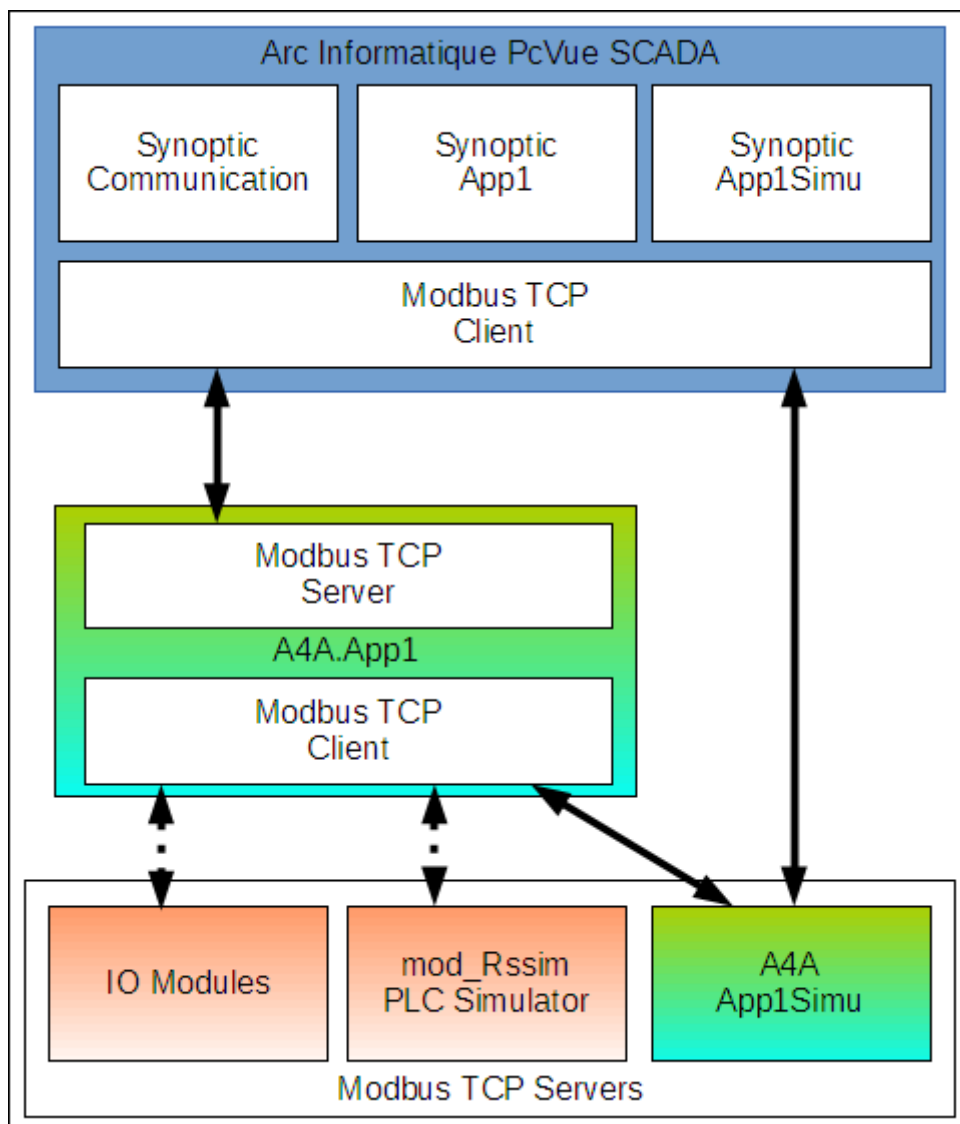


Figure 8.1: app1 Overview

8.3 Organic specifications

After the functional specifications, what the system should do, we can specify how it can do it.

So we brought all the information, sensors and actuators in the control box and all this was wired on a remote inputs / outputs module, Modbus TCP server.

8.3.1 Assignment of Inputs / Outputs

Agree that the supplied module allows:

- access to the status of digital inputs wired reading the "Discrete Inputs"
- to control the cabled output by writing "Coils", which can also be read,
- to read the analog measurements by reading "Input Registers".

Table 8.1: Digital inputs, from the "App1" application's point of view

No	Mnemonic	Description
0	Auto	Auto position switch
1	Manu	Manu position switch
2	Ack_Faults	Push button Faults Acknowledge
3	Level_Switch_11	Level Switch High
4	Valve_14_Pos_Open	Valve 14, position open
5	Valve_14_Pos_Closed	Valve 14, position closed
6	Level_Switch_12	Level Switch Low
7	MyPump13_FeedBack	Pump 13, contactor feedback

Table 8.2: Digital outputs, from the "App1" application's point of view

No	Mnemonic	Description
0	MyPump13_Coil	Pump 13, contactor coil
1	Valve_14_Coil	Valve 14, coil

Table 8.3: Analog inputs, from the "App1" application's point of view

No	Mnemonic	Description
0	Level_Transmitter_10_Measure	Level Measure LT10

8.4 The app1 project

As mentioned a little further upstream:

GNAT Pro Studio allows to arrange the projects in a project tree with inheriting other projects. This will expand the project by adding or substituting source code files. You will please refer to GPRbuild documentation regarding the projects.

A4A/app1.gpr This is the application example 1 project file, it extends the main project "a4a.gpr". Only the user program differs from the one included in the main project.

So each file listed in the sources folder, "app1/src", and bears the same name as the one that exists in the sources of the main project, in the "src" folder, replaces or hides the main project file.

While a file existing only in the "app1/src" folder extends the main project.

In what follows, we strive to show what determines the logic of this app1 application.

8.5 Memory areas

The process image of inputs and outputs is defined in package "A4A.Memory" in the "src" directory.

If the size is not enough, change it accordingly to the needs.

```

package A4A.Memory is
    -----
    -- IO Areas
    -----

end A4A.Memory;

package A4A.Memory.MBTCP_IOServer is
    -----
    -- IO Areas
    -----

    Bool_IO_Size : constant := 1024;
    subtype Bool_IO_Range is Integer range 0 .. Bool_IO_Size - 1;

    Bool_Inputs : Bool_Array (Bool_IO_Range) := (others => False);
    Bool_Coils : Bool_Array (Bool_IO_Range) := (others => False);

    Word_IO_Size : constant := 1024;
    subtype Word_IO_Range is Integer range 0 .. Word_IO_Size - 1;

    Input_Registers : Word_Array (Word_IO_Range) := (others => 0);
    Registers : Word_Array (Word_IO_Range) := (others => 0);

end A4A.Memory.MBTCP_IOServer;

package A4A.Memory.MBTCP_IOScan is
    -----
    -- IO Areas
    -----

    Bool_IO_Size : constant := 1024;
    subtype Bool_IO_Range is Integer range 0 .. Bool_IO_Size - 1;

    Bool_Inputs : Bool_Array (Bool_IO_Range) := (others => False);
    Bool_Outputs : Bool_Array (Bool_IO_Range) := (others => False);

    Word_IO_Size : constant := 1024;
    subtype Word_IO_Range is Integer range 0 .. Word_IO_Size - 1;

    Word_Inputs : Word_Array (Word_IO_Range) := (others => 0);
    Word_Outputs : Word_Array (Word_IO_Range) := (others => 0);

end A4A.Memory.MBTCP_IOScan;

```

8.6 Modbus TCP IO Scanning Configuration

A detailed description of this scan is available in Chapter [Design](#).

The configuration takes place in the file "a4a-application-mbtcp_clients_config.ads" in folder "app1/src".

For each Modbus TCP server, we use a task that will be created automatically from the configuration entered here.

So for each Modbus TCP server or I/O module, create a configuration and insert it in configurations table "MBTCP_Clients_Configuration".

There! It is done.

The following configuration declares two servers on the same PC, the one of the simulation and the one of the application app1 itself. As the latter loopback is used only for testing purposes, it is disabled.

Note also that only commands 1-3 are useful in this application, the others are there for demonstration and testing.

```
with A4A.MBTCP_Client; use A4A.MBTCP_Client;
with A4A.Protocols; use A4A.Protocols.IP_Address_Strings;
with A4A.Protocols.LibModbus; use A4A.Protocols.LibModbus;

package A4A.Application.MBTCP_Clients_Config is

-----
-- Modbus TCP Clients configuration
-----

-- For each Modbus TCP Server define one client configuration task

Config1 : aliased Client_Configuration :=
  (Command_Number   => 11,
   Enabled          => True,
   Debug_On        => False,
   Task_Period_MS  => 10,
   Retries         => 3,
   Timeout         => 0.2,

   Server_IP_Address => To_Bounded_String("192.168.0.100"),
   -- 127.0.0.1 / 192.168.0.100

   Server_TCP_Port  => 502,
   -- 502 Standard / 1502 PIC Simu / 1504 App1Simu

   Commands =>
     (
      --
      --
      -- Action Enabled Multiple Shift Number Remote Local
      1 =>
        (Read_Input_Registers, True, 10, 0, 10, 0, 0),
      2 =>
        ( Read_Registers, True, 20, 0, 10, 0, 10),
      3 =>
        ( Write_Registers, True, 30, 0, 20, 20, 0),
      4 =>
        ( Read_Bits, True, 30, 1, 16, 0, 0),
      5 =>
        ( Read_Input_Bits, True, 30, 2, 16, 0, 32),
      6 =>
        ( Write_Register, True, 30, 3, 1, 50, 30),
      7 =>
        ( Write_Bit, True, 30, 4, 1, 0, 0),
      8 =>
        ( Write_Bits, True, 30, 5, 15, 1, 1),

      9 => (Action => Write_Read_Registers,
```

```

        Enabled           => True,
        Period_Multiple  => 10,
        Shift            => 5,
        Write_Number     => 10,
        Write_Offset_Remote => 40,
        Write_Offset_Local  => 20,
        Read_Number      => 10,
        Read_Offset_Remote => 10,
        Read_Offset_Local => 20),
    10 =>
    (   Read_Registers,  True,    50,    0,    10,   100,   100),
    11 =>
    (   Read_Registers,  True,    50,    1,    10,   110,   110)
)
);

Config2 : aliased Client_Configuration :=
(Command_Number   => 2,
 Enabled         => False,
 Debug_On       => False,
 Task_Period_MS => 100,
 Retries        => 3,
 Timeout        => 0.2,

 Server_IP_Address => To_Bounded_String("127.0.0.1"),
 Server_TCP_Port   => 1503, -- My own MBTCP server

 Commands =>
  ( --
    --
    -- Action Enabled Multiple Shift Number Remote Local
    1 =>
    (   Read_Registers,  True,    10,    0,    10,    0,    0),
    2 =>
    (   Write_Registers, True,    30,    1,    10,    0,    0)
  )
);

-- Declare all clients configuration in the array
-- The kernel will create those clients accordingly

MBTCP_Clients_Configuration : Client_Configuration_Access_Array :=
  (1 => Config1'Access,
   2 => Config2'Access);

end A4A.Application.MBTCP_Clients_Config;

```

In the kitchens of "Ada for Automation", Modbus TCP client tasks are created.

They communicate with the rest of the system through an emulation of dual port memory that helps ensure data consistency.

Data is exchanged with the image of the inputs and outputs, so two boolean and two registers arrays.

The curious can go see the package "A4A.Kernel" in the "src" folder.

8.7 Modbus TCP Server Configuration

This configuration takes place in the file "a4a-application-mbtcp_server_config.ads" in folder "app1/src".

We instantiate the generic package "A4A.MBTCP_Server" by providing the desired quantities of items and assign a configuration.

```

with A4A.MBTCP_Server;
with A4A.Protocols; use A4A.Protocols.IP_Address_Strings;

package A4A.Application.MBTCP_Server_Config is

-----
-- Modbus TCP Server configuration
-----

package Server is new A4A.MBTCP_Server
(
  Coils_Number           => 65536,
  Input_Bits_Number      => 65536,
  Input_Registers_Number => 65536,
  Registers_Number       => 65536
);

Config1 : aliased Server.Server_Configuration :=
(Server_Enabled           => True,
 Debug_On                 => False,
 Retries                  => 3,
 Server_IP_Address        => To_Bounded_String("127.0.0.1"),
 Server_TCP_Port          => 1502); -- 1503

end A4A.Application.MBTCP_Server_Config;

```

The corresponding memory areas were of course defined, cf. above the package "A4A.Memory".

8.8 User objects

So the inputs and outputs are connected to the process image of the inputs and outputs.

With a conventional controller, you would define a variable table to play with your sensors and actuators during the synchronization phase for example.

Such a concept does not exist for the moment in "Ada for Automation". This is not very serious, you can use the Modbus TCP server that has the I / O data. You will just have to create either synoptics of your equipment or a simple table ...

You also have the GNAT Pro Studio debugger (GDB).

The next step is the definition of objects in your application.

In this example application, this is achieved in package "A4A.User_Objects".

This is by no means an obligation and you can structure your application as you see fit.

So the following code declares inputs, outputs, internal variables, object instances.

It is altogether equivalent to what you would do in a PLC program without input screens.

Nothing more than text files, some readers have already left?

```

with A4A.Library.Timers.TON; use A4A.Library.Timers;
with A4A.Library.Devices.Contactor; use A4A.Library.Devices;
with A4A.Library.Devices.Alarm_Switch; use A4A.Library.Devices;
with A4A.Library.Devices.Valve; use A4A.Library.Devices;
with A4A.Library.Analog.PID; use A4A.Library.Analog;
with A4A.Library.Analog.Threshold; use A4A.Library.Analog;

package A4A.User_Objects is

```

```
-- User Objects creation
-----

-- Inputs
-----

Auto          : Boolean := False;
Manu          : Boolean := False;

Ack_Faults    : Boolean := False;
-- Faults Acknowledgement

Level_Transmitter_10_Measure : Word    := 0;

Level_Switch_11 : Boolean := False;
Level_Switch_12 : Boolean := False;
MyPump13_FeedBack : Boolean := False;
Valve_14_Pos_Open : Boolean := False;
Valve_14_Pos_Closed : Boolean := False;

-----

-- Outputs
-----

MyPump13_Coil : Boolean := False;
Valve_14_Coil : Boolean := False;

-----

-- HMI Inputs
-----

MyPump13_Manu_Cmd_On : Boolean := False;
Valve_14_Manu_Cmd_Open : Boolean := False;

-----

-- Internal
-----

Tempo_TON_1 : TON.Instance;
-- My Tempo TON 1

Tempo_TON_2 : TON.Instance;
-- My Tempo TON 2

TON_2_Q : Boolean := False;

Mode_Auto : Boolean := False;
-- Working Mode is Auto

Mode_Manu : Boolean := False;
-- Working Mode is Manu

Level_Transmitter_10_Value : Float := 0.0;
Level_Transmitter_10_Hyst : Float := 5.0;
Level_Transmitter_10_HHH_T : Float := 99.0;
Level_Transmitter_10_HH_T : Float := 90.0;
Level_Transmitter_10_H_T : Float := 85.0;
Level_Transmitter_10_L_T : Float := 15.0;
Level_Transmitter_10_LL_T : Float := 10.0;
Level_Transmitter_10_LLL_T : Float := 1.0;

Level_Transmitter_10_InitDone : Boolean := False;
```



```

Level_Transmitter_10_XHHH : Boolean := False;
Level_Transmitter_10_XHH  : Boolean := False;
Level_Transmitter_10_XH   : Boolean := False;
Level_Transmitter_10_XL   : Boolean := False;
Level_Transmitter_10_XLL  : Boolean := False;
Level_Transmitter_10_XLLL : Boolean := False;

Valve_14_Condition_Perm  : Boolean := False;
Valve_14_Condition_Auto : Boolean := False;
Valve_14_Condition_Manu : Boolean := False;

Valve_14_Auto_Cmd_Open  : Boolean := False;

Valve_14_Cmd_Open: Boolean := False;
-- Valve Command

MyPump13_Condition_Perm  : Boolean := False;
MyPump13_Condition_Auto  : Boolean := False;
MyPump13_Condition_Manu  : Boolean := False;

MyPump13_Auto_Cmd_On    : Boolean := False;

MyPump13_Cmd_On        : Boolean := False;
-- Pump Command

MyPump13_Is_On         : Boolean := False;
-- Pump Status

My_PID_1               : PID.Instance;
-- My PID Controller 1

Level_Transmitter_10_Thresholds : Threshold.Instance;
-- Level_Transmitter_10 Thresholds Box

-----
-- Devices Instances
-----

MyPump13                : Contactor.Instance :=
  Contactor.Create (Id => "Pump13");
-- Pump Instance

LS11_AH                 : Alarm_Switch.Instance :=
  Alarm_Switch.Create
    (Id           => "LS11",
     TON_Preset => 2000);
-- Level Alarm Switch Instance

LS12_AL                 : Alarm_Switch.Instance :=
  Alarm_Switch.Create
    (Id           => "LS12",
     TON_Preset => 2000);
-- Level Alarm Switch Instance

Valve_14                : Valve.Instance :=
  Valve.Create
    (Id           => "XV14",
     TON_Preset => 5000); -- a slow valve
-- Valve Instance

```

```
end A4A.User_Objects;
```

8.9 I/O Mapping

There must be matching your input and output objects with the process image.

This is done on your initiative in user functions you organize as you think.

Such as:

```
with A4A.Memory.MBTCP_IOServer;
with A4A.Memory.MBTCP_IOScan;
use A4A.Memory;

with A4A.Library.Conversion; use A4A.Library.Conversion;

with A4A.User_Objects; use A4A.User_Objects;

package body A4A.User_Functions is

    -----
    -- User functions
    -----

    procedure Map_Inputs is
    --      Temp_Bools : array (0..15) of Boolean := (others => False);
    begin

        Word_To_Booleans
        --      (Word_in      => MBTCP_IOScan.Word_Inputs (0),
        --      Boolean_out00 => Auto,
        --      Boolean_out01 => Manu,
        --      Boolean_out02 => Ack_Faults,
        --      Boolean_out03 => Level_Switch_11,
        --      Boolean_out04 => Valve_14_Pos_Open,
        --      Boolean_out05 => Valve_14_Pos_Closed,
        --      Boolean_out06 => Level_Switch_12,
        --      Boolean_out07 => MyPump13_FeedBack,
        --
        --      Boolean_out08 => Temp_Bools (8) , -- Spare
        --      Boolean_out09 => Temp_Bools (9) , -- Spare
        --      Boolean_out10 => Temp_Bools (10), -- Spare
        --      Boolean_out11 => Temp_Bools (11), -- Spare
        --      Boolean_out12 => Temp_Bools (12), -- Spare
        --      Boolean_out13 => Temp_Bools (13), -- Spare
        --      Boolean_out14 => Temp_Bools (14), -- Spare
        --      Boolean_out15 => Temp_Bools (15) -- Spare
        --      );

        Auto           := MBTCP_IOScan.Bool_Inputs (32);
        Manu           := MBTCP_IOScan.Bool_Inputs (33);
        Ack_Faults    := MBTCP_IOScan.Bool_Inputs (34);
        Level_Switch_11 := MBTCP_IOScan.Bool_Inputs (35);
        Valve_14_Pos_Open := MBTCP_IOScan.Bool_Inputs (36);
        Valve_14_Pos_Closed := MBTCP_IOScan.Bool_Inputs (37);
        Level_Switch_12 := MBTCP_IOScan.Bool_Inputs (38);
        MyPump13_FeedBack := MBTCP_IOScan.Bool_Inputs (39);

        Level_Transmitter_10_Measure := MBTCP_IOScan.Word_Inputs (0);
```

```
end Map_Inputs;

procedure Map_Outputs is
begin
    --      Booleans_To_Word
    --      (Boolean_in00 => MyPump13_Coil,
    --      Boolean_in01 => Valve_14_Coil,
    --      -- others => Spare
    --      Word_out      => MBTCP_IOScan.Word_Outputs (0)
    --      );

    MBTCP_IOScan.Bool_Outputs (0) := MyPump13_Coil;
    MBTCP_IOScan.Bool_Outputs (1) := Valve_14_Coil;

end Map_Outputs;

procedure Map_HMI_Inputs is
--      Temp_Bools : array (0..15) of Boolean := (others => False);
begin
    --      Word_To_Booleans
    --      (Word_in      => MBTCP_IOServer.Registers (0),
    --      Boolean_out00 => MyPump13_Manu_Cmd_On,
    --      Boolean_out01 => Valve_14_Manu_Cmd_Open,
    --      Boolean_out02 => Temp_Bools (2) , -- Spare
    --      Boolean_out03 => Temp_Bools (3) , -- Spare
    --      Boolean_out04 => Temp_Bools (4) , -- Spare
    --      Boolean_out05 => Temp_Bools (5) , -- Spare
    --      Boolean_out06 => Temp_Bools (6) , -- Spare
    --      Boolean_out07 => Temp_Bools (7) , -- Spare
    --
    --      Boolean_out08 => Temp_Bools (8) , -- Spare
    --      Boolean_out09 => Temp_Bools (9) , -- Spare
    --      Boolean_out10 => Temp_Bools (10), -- Spare
    --      Boolean_out11 => Temp_Bools (11), -- Spare
    --      Boolean_out12 => Temp_Bools (12), -- Spare
    --      Boolean_out13 => Temp_Bools (13), -- Spare
    --      Boolean_out14 => Temp_Bools (14), -- Spare
    --      Boolean_out15 => Temp_Bools (15) -- Spare
    --      );

    MyPump13_Manu_Cmd_On := MBTCP_IOServer.Bool_Coils (0);
    Valve_14_Manu_Cmd_Open := MBTCP_IOServer.Bool_Coils (1);

end Map_HMI_Inputs;

procedure Map_HMI_Outputs is
begin
    --      Booleans_To_Word
    --      (Boolean_in00 => MyPump13.is_On,
    --      Boolean_in01 => MyPump13.is_Faulty,
    --      Boolean_in02 => Valve_14.is_Open,
    --      Boolean_in03 => Valve_14.is_Closed,
    --      Boolean_in04 => Valve_14.is_Faulty,
    --      Boolean_in05 => LS11_AH.is_On,
    --      Boolean_in06 => LS11_AH.is_Faulty,
    --      Boolean_in07 => LS12_AL.is_On,
    --
    --      Boolean_in08 => LS12_AL.is_Faulty,
    --      -- others => Spare
```

```

--      Word_out      => MBTCP_IOServer.Input_Registers(0)
--      );

MBTCP_IOServer.Bool_Inputs (0) := MyPump13.is_On;
MBTCP_IOServer.Bool_Inputs (1) := MyPump13.is_Faulty;
MBTCP_IOServer.Bool_Inputs (2) := Valve_14.is_Open;
MBTCP_IOServer.Bool_Inputs (3) := Valve_14.is_Closed;
MBTCP_IOServer.Bool_Inputs (4) := Valve_14.is_Faulty;
MBTCP_IOServer.Bool_Inputs (5) := LS11_AH.is_On;
MBTCP_IOServer.Bool_Inputs (6) := LS11_AH.is_Faulty;
MBTCP_IOServer.Bool_Inputs (7) := LS12_AL.is_On;
MBTCP_IOServer.Bool_Inputs (8) := LS12_AL.is_Faulty;

MBTCP_IOServer.Input_Registers(0) := Level_Transmitter_10_Measure;

end Map_HMI_Outputs;
end A4A.User_Functions;

```

8.10 Main procedure

The main loop in "A4A.Kernel" calls the procedure "Main_Cyclic" from package "A4A.Application".

This is fairly standard in automation, agree on.

There we find the procedures defined above: "Map_Inputs" and "Map_Outputs".

These procedures without parameters and without parentheses are very similar to subroutines.

```

...
package body A4A.Application is
...

  procedure Main_Cyclic is
    My_Ident : String := "A4A.Application.Main_Cyclic";
    Elapsed_TON_2 : Ada.Real_Time.Time_Span;
  --      Bug : Integer := 10;
  begin
    A4A.Log.Logger.Put (Who => My_Ident,
  --      What => "Yop ! ***** "
  --      & Integer'Image(Integer(MBTCP_IOScan_Inputs(0))));

  --      Bug := Bug / (Bug - 10);
    Map_Inputs;

    Map_HMI_Inputs;

  -- Working mode
    Mode_Auto := Auto and not Manu;
    Mode_Manu := not Auto and Manu;

  -- Level Transmitter 10
    Level_Transmitter_10_Value := Scale_In
      (X      => Integer(Level_Transmitter_10_Measure),
       Xmin => 0,
       Xmax => 65535,
       Ymin => 0.0,
       Ymax => 100.0);

    if not Level_Transmitter_10_InitDone then
      Level_Transmitter_10_Thresholds.Initialise

```

```

    (Hysteresis => Level_Transmitter_10_Hyst,
     HHH_T     => Level_Transmitter_10_HHH_T,
     HH_T      => Level_Transmitter_10_HH_T,
     H_T       => Level_Transmitter_10_H_T,
     L_T       => Level_Transmitter_10_L_T,
     LL_T      => Level_Transmitter_10_LL_T,
     LLL_T     => Level_Transmitter_10_LLL_T);

    Level_Transmitter_10_InitDone := True;
end if;

Level_Transmitter_10_Thresholds.Cyclic
(Value => Level_Transmitter_10_Value,
 HHH  => Level_Transmitter_10_XHHH,
 HH   => Level_Transmitter_10_XHH,
 H    => Level_Transmitter_10_XH,
 L    => Level_Transmitter_10_XL,
 LL   => Level_Transmitter_10_XLL,
 LLL  => Level_Transmitter_10_XLLL);

LS11_AH.Cyclic(Alarm_Cond => not Level_Switch_11,
               Ack         => Ack_Faults,
               Inhibit    => False);

LS12_AL.Cyclic(Alarm_Cond => not Level_Switch_12,
               Ack         => Ack_Faults,
               Inhibit    => False);

-- Valve_14 Command
Valve_14_Condition_Perm := not LS11_AH.is_Faulty;
Valve_14_Condition_Auto := Mode_Auto;
Valve_14_Condition_Manu := Mode_Manu;

if Level_Transmitter_10_XL then
    Valve_14_Auto_Cmd_Open := True;
elsif Level_Transmitter_10_XHH then
    Valve_14_Auto_Cmd_Open := False;
end if;

Valve_14_Cmd_Open := Valve_14_Condition_Perm and
((Valve_14_Condition_Manu and Valve_14_Manu_Cmd_Open)
 or (Valve_14_Condition_Auto and Valve_14_Auto_Cmd_Open));

Valve_14.Cyclic (Pos_Open  => Valve_14_Pos_Open,
                 Pos_Closed => Valve_14_Pos_Closed,
                 Ack       => Ack_Faults,
                 Cmd_Open  => Valve_14_Cmd_Open,
                 Coil      => Valve_14_Coil);

-- MyPump13 Command
MyPump13_Condition_Perm := not LS12_AL.is_Faulty;
MyPump13_Condition_Auto := Mode_Auto;
MyPump13_Condition_Manu := Mode_Manu;

if Level_Transmitter_10_XHH then
    MyPump13_Auto_Cmd_On := True;
elsif Level_Transmitter_10_XLL then
    MyPump13_Auto_Cmd_On := False;
end if;

MyPump13_Cmd_On := MyPump13_Condition_Perm and
((MyPump13_Condition_Auto and MyPump13_Auto_Cmd_On)

```

```

    or (MyPump13_Condition_Manu and MyPump13_Manu_Cmd_On));

MyPump13.Cyclic(Feed_Back => MyPump13_FeedBack,
                Ack        => Ack_Faults,
                Cmd_On     => MyPump13_Cmd_On,
                Coil       => MyPump13_Coil);

-- Status use example
MyPump13_Is_On := MyPump13.is_On;

--      A4A.Log.Logger.Put (Who => My_Ident,
--                          What => "Here is MyPump Id : " & MyPump13.Get_Id);
--
-- A little test
Tempo_TON_2.Cyclic (Start  => not TON_2_Q,
                    Preset  => Ada.Real_Time.Milliseconds (10000),
                    Elapsed => Elapsed_TON_2,
                    Q       => TON_2_Q);

if TON_2_Q then
    A4A.Log.Logger.Put (Who => My_Ident,
                      What => "Tempo_TON_2 elapsed!");
end if;

-- Modbus TCP IO Scanning test
for Index in 10 .. 19 loop
    A4A.Memory.MBTCP_IOScan.Word_Outputs (Index) := Word (Index);
end loop;

for Index in 20 .. 29 loop
    A4A.Memory.MBTCP_IOScan.Word_Outputs (Index) :=
        A4A.Memory.MBTCP_IOScan.Word_Inputs (Index);
end loop;

A4A.Memory.MBTCP_IOScan.Word_Outputs (30) :=
    A4A.Memory.MBTCP_IOScan.Word_Outputs (30) + 1;

A4A.Memory.MBTCP_IOScan.Bool_Outputs (0 .. 15) :=
    A4A.Memory.MBTCP_IOScan.Bool_Inputs (32 .. 47);

-- Modbus TCP Server test
A4A.Memory.MBTCP_IOServer.Input_Registers (5 .. 19) :=
    A4A.Memory.MBTCP_IOServer.Registers (5 .. 19);

Map_Outputs;

Map_HMI_Outputs;

exception -- ❶

when Error: others =>
    A4A.Log.Logger.Put (Who => My_Ident,
                      What => Exception_Information(Error));

    Program_Fault_Flag := True;

end Main_Cyclic;

...
end A4A.Application;
...

```

- If case of programming error, such as division by 0, an exception is thrown and the error information is traced in the log while the **Program_Fault_Flag** flag is waving.

Will easily admit that the code presented above reads as easily as "Structured Text".

8.11 Kernel loop

The main loop that handles everything, cf. "task body Main_Task" in "A4A.Kernel.Main" is presented below in simplified version to illustrate how everything revolves.

Of course you can modify it to your liking, like the rest of the code of this project, but it is not absolutely necessary and you can probably create many applications with this code.

You can find in it:

- the Modbus TCP server initialization,
- the initialization of the Modbus TCP clients,
- the organization of the mentioned data streams [here](#).

```

...
package body A4A.Kernel.Main is
...

begin

  Log_Task_Start;

  -----
  -- Modbus TCP Server Management
  -----

  MBTCP_Server_Task := new Server.Periodic_Task
    (Task_Priority => System.Default_Priority,
     Configuration => A4A.Application.MBTCP_Server_Config.Config1'Access
    );

  A4A.Log.Logger.Put (Who => My_Ident,
                    What => "Modbus TCP Server created...");

  -----
  -- Modbus TCP Clients Management
  -----

  for Index in MBTCP_Clients_Tasks'Range loop
    MBTCP_Clients_Tasks(Index) := new A4A.MBTCP_Client.Periodic_Task
      (Task_Priority => System.Default_Priority,
       Configuration => MBTCP_Clients_Configuration(Index),
       DPM_Access   => My_DPM'Access);
  end loop;

  A4A.Log.Logger.Put (Who => My_Ident,
                    What => "Modbus TCP Clients created...");

  -----
  -- Main loop
  -----

  loop

```

```
A4A.Memory.MBTCP_IOScan_Inputs := My_DPM.Inputs.Get_Data
  (Offset => A4A.Memory.MBTCP_IOScan_Inputs'First,
   Number => A4A.Memory.MBTCP_IOScan_Inputs'Length);

Server.Registers_Read
  (Outputs => MBTCP_IOServer_Registers,
   Offset  => 0);

A4A.Application.Main_Cyclic;

My_DPM.Outputs.Set_Data
  (Data_In => A4A.Memory.MBTCP_IOScan_Outputs,
   Offset  => A4A.Memory.MBTCP_IOScan_Outputs'First);

Server.Inputs_Registers_Write
  (Inputs => A4A.Memory.MBTCP_IOScan_Inputs,
   Offset => 0);

Server.Inputs_Registers_Write
  (Inputs => A4A.Memory.MBTCP_IOScan_Outputs,
   Offset => A4A.Memory.MBTCP_IOScan_Inputs'Length);

Server.Inputs_Registers_Write
  (Inputs => MBTCP_IOServer_Input_Registers,
   Offset => (A4A.Memory.MBTCP_IOScan_Inputs'Length
             + A4A.Memory.MBTCP_IOScan_Outputs'Length));

exit when Quit;

case Application_Main_Task_Type is
when Cyclic =>
  delay My_Delay;
when Periodic =>
  Next_Time := Next_Time + My_Period;
  delay until Next_Time;
end case;
end loop;

...
end A4A.Kernel.Main;
...
```


Chapter 9

Hilscher

Hilscher GmbH is a German company that was founded in 1986 and develops communication products for over twenty years. Hilscher designs and produces industrial communication solutions from the "System on Chip - SoC" to the complete equipment. We will briefly present here only those products designed around the new generation of netX processor.

9.1 Components

9.1.1 The netX family

Hilscher netX are a family of "System on Chip", integrating around an ARM processor a number of peripherals including one or more communication channels.

These communication channels have the distinction of receiving their function by micro-code and are therefore much more flexible than if they were ... well ... frozen ... like ASICs.

Thus netX can be used to manage the entire range of market protocols.

The netX are designed by Hilscher SoC, a subsidiary of Hilscher GmbH in Berlin. Hilscher SoC can also design your chip based on your specifications.

The netX are hard hit because they are the basis of all Hilscher derived products, used in many industrial applications.

You can also use the netX in your own products, according to the terms of a contract to sign with the company.

9.1.2 The rcX Real Time Operating System

For the netX family Hilscher has developed a real-time operating system optimized for industrial communication, and royalty free.

This operating system includes the necessary drivers for the use of the various components of the netX.

The rcX system can be used on any hardware platform based on the netX.

It is possible to develop an own platform or use any of the devices offered by Hilscher. For example you could create a specific firmware for a netTAP gateway managing your own protocol.

9.1.3 Hardware abstraction layers

The HAL, Hardware Abstraction Layer, define the interface between drivers and hardware. By providing these layers, Hilscher allows porting other operating systems on the netX and developing your own protocol stacks.

To date, the BSP (Board Support Package) for Linux and other OS are available.

9.1.4 Protocol stacks

Hilscher is a member of various organizations which govern industrial communication protocols and participates in the development of standards.

Hilscher has developed a broad portfolio of protocols. Whether legacy as Profibus, DeviceNet, CANopen, AS-i, CC-Link, or real-time Ethernet-based, such as Profinet, Sercos III, EtherCAT, PowerLink, Ethernet/IP or Open Modbus TCP, in Master or Slave version, all industry standard protocols are available on the netX.

9.2 Products designed around the netX

Communication needs are extremely diverse and Hilscher has created a range of products covering the needs of the hardware developer, of the automation engineer, of the computer programmer at the OEM, of the integrator or assembler.

So we find solutions for the embedded world to enable electronic equipments such as drives, encoders, measuring instruments, operator panels, etc . . . to communicate using proven components.

In the PC world, Hilscher offers a range of boards in all formats. These boards are used for test and measurement benches or advanced automation solutions with integrated PLC complying with IEC 61131-3 or even IEC 61499.

The gateway range allows integration of various devices, from a revamping process or of particular interest for the application, in a network architecture using a protocol unsupported by those devices.

Finally, the activity "standard products" is completed in a high proportion of OEM (Original Equipment Manufacturer).

9.3 Configuration tools

To configure protocol stacks, Hilscher has developed SYCON.net, a tool based on FDT / DTM technology.

This software allows to configure all the products in the range built around the netX.

A lighter tool was developed, the netX Configuration Tool for configuring in a simple way the products in slave version, configuration of slave protocol stacks being significantly reduced compared to master versions.

9.4 Drivers

For PC cards, cifX, Hilscher has developed drivers for several operating systems of the market.

These drivers are based on a ToolKit in C, whose source code is provided, which allows to port the driver to an operating system not supported by Hilscher.

Regardless of the hardware platform, the chosen operating system or protocol used, the interface is common. Your application has a maximum connectivity with minimal porting work.

There is an emulation of this driver for embedded development, allowing to begin the development on a PC-based equipment, enjoying the comfort of this platform, and port without much effort the project on the embedded equipment.

Above these generic drivers, specific drivers have been developed for the major software PLC of the market.

Note

Hilscher GmbH works with companies including 3S, which develops CoDeSys, KW Software with ProConOS, IBH Softec offering solutions compatible with Siemens.

At Hilscher France, for example, we developed the ISaGRAF driver for all supported operating systems and we have also ported ISaGRAF on the netX.

We have also developed the driver for WinAC / RTX PLC Siemens, driver that has since been taken over by our parent company.

Chapter 10

Application Example 2

This application implements a Hilscher PROFIBUS DP Master card cifX and a Modbus TCP server.

Cf. : <http://slo-ist.fr/hilscher/cifx/a4a-exemple-dapplication-app2-hilscher-cifx>

Chapter 11

Application Example 3

This application implements a Modbus TCP server and a Modbus RTU Master.

Cf. : <http://slo-ist.fr/ada4automation/a4a-exemple-dapplication-app3-modbus-rtu-maitre>

Chapter 12

Library

We discuss the library of "Ada for Automation" including basic data types, functions, procedures and useful objects in an automation application.

A "Ada for Automation" application benefits from this environment but also the plethora of useful libraries available in Ada, written in Ada or C with a binding, as libmodbus or GtkAda.

12.1 Basic types

Ada defines a number of basic types and provides the means to create own data types. This is not the subject of this book.

In the parent package A4A, can be found some defined basic types and functions that the control engineer can use in all of children packages without having to specify "with" or "use" clauses.

12.1.1 Types

```
with Interfaces;

with Ada.Text_IO;
with Ada.Characters.Latin_1; use Ada.Characters.Latin_1;

with Ada.Unchecked_Conversion;

package A4A is

    -----
    -- Elementary types
    -----

    type Byte is new Interfaces.Unsigned_8;
    -- 8-bit unsigned integer

    type Word is new Interfaces.Unsigned_16;
    -- 16-bit unsigned integer

    type DWord is new Interfaces.Unsigned_32;
    -- 32-bit unsigned integer

    type LWord is new Interfaces.Unsigned_64;
    -- 64-bit unsigned integer

    type SInt is new Interfaces.Integer_8;
```

```

-- 8-bit signed integer

type Int is new Interfaces.Integer_16;
-- 16-bit signed integer

type DInt is new Interfaces.Integer_32;
-- 32-bit signed integer

type LInt is new Interfaces.Integer_64;
-- 64-bit signed integer

```

12.1.2 Shifting and rotation

There are the elementary functions for shifting and rotation, to the right or to the left, and for each or almost of the elementary types

```

-----
-- Elementary functions
-----

function SHL
  (Value : Byte;
   Amount : Natural) return Byte renames Shift_Left;
-- <summary>Shift Left Byte</summary>

function SHR
  (Value : Byte;
   Amount : Natural) return Byte renames Shift_Right;
-- <summary>Shift Right Byte</summary>

function SHL
  (Value : Word;
   Amount : Natural) return Word renames Shift_Left;
-- <summary>Shift Left Word</summary>

function SHR
  (Value : Word;
   Amount : Natural) return Word renames Shift_Right;
-- <summary>Shift Right Word</summary>

function SHL
  (Value : DWord;
   Amount : Natural) return DWord renames Shift_Left;
-- <summary>Shift Left DWord</summary>

function SHR
  (Value : DWord;
   Amount : Natural) return DWord renames Shift_Right;
-- <summary>Shift Right DWord</summary>

function ROL
  (Value : Byte;
   Amount : Natural) return Byte renames Rotate_Left;
-- <summary>Rotate Left Byte</summary>

function ROR
  (Value : Byte;
   Amount : Natural) return Byte renames Rotate_Right;
-- <summary>Rotate Right Byte</summary>

```

```

function ROL
  (Value : Word;
   Amount : Natural) return Word renames Rotate_Left;
-- <summary>Rotate Left Word</summary>

function ROR
  (Value : Word;
   Amount : Natural) return Word renames Rotate_Right;
-- <summary>Rotate Right Word</summary>

function ROL
  (Value : DWord;
   Amount : Natural) return DWord renames Rotate_Left;
-- <summary>Rotate Left DWord</summary>

function ROR
  (Value : DWord;
   Amount : Natural) return DWord renames Rotate_Right;
-- <summary>Rotate Right DWord</summary>

```

12.1.3 Bytes and words arrays

Also there are the unconstrained array types compatible with function calls in C:

```

-----
-- Unconstrained Arrays of Elementary types
-----

type Byte_Array is array (Integer range <>) of aliased Byte;
for Byte_Array'Component_Size use Byte' Size;
pragma Pack(Byte_Array);
pragma Convention (C, Byte_Array);
type Byte_Array_Access is access all Byte_Array;

type Word_Array is array (Integer range <>) of aliased Word;
for Word_Array'Component_Size use Word' Size;
pragma Pack(Word_Array);
pragma Convention (C, Word_Array);
type Word_Array_Access is access all Word_Array;

```

12.1.4 Text_IO

To facilitate the use of the console to debug and trace, we also provide instances of the Text_IO generic package for each elementary type:

```

-----
-- Instanciation of Generic Text_IO package for each Elementary type
-----

package Byte_Text_IO is
  new Ada.Text_IO.Modular_IO (Byte);

package Word_Text_IO is
  new Ada.Text_IO.Modular_IO (Word);

package DWord_Text_IO is
  new Ada.Text_IO.Modular_IO (DWord);

```

```
package SInt_Text_IO is
  new Ada.Text_IO.Integer_IO (SInt);

package Int_Text_IO is
  new Ada.Text_IO.Integer_IO (Int);

package DInt_Text_IO is
  new Ada.Text_IO.Integer_IO (DInt);
```

12.1.5 Unchecked conversions

Ada is a language which is really picky on types and we cannot do a weird thing without indicating that this is really our intention. When one wants to convert one type to another and this can be a problem, it is necessary to do so explicitly, hence the following conversions functions, so use with care:

```
function SInt_To_Byte is new Ada.Unchecked_Conversion
(Source => SInt,
 Target => Byte);

function Byte_To_SInt is new Ada.Unchecked_Conversion
(Source => Byte,
 Target => SInt);

function Int_To_Word is new Ada.Unchecked_Conversion
(Source => Int,
 Target => Word);

function Word_To_Int is new Ada.Unchecked_Conversion
(Source => Word,
 Target => Int);

function DInt_To_DWord is new Ada.Unchecked_Conversion
(Source => DInt,
 Target => DWord);

function DWord_To_DInt is new Ada.Unchecked_Conversion
(Source => DWord,
 Target => DInt);

function LInt_To_LWord is new Ada.Unchecked_Conversion
(Source => LInt,
 Target => LWord);

function LWord_To_LInt is new Ada.Unchecked_Conversion
(Source => LWord,
 Target => LInt);
```

12.2 Conversions

In automation as industrial computing, we spend time gathering Boolean information in bytes or words, 16, 32 or 64 bits because it is easier to process or vice versa easier to transport, and of course the opposite.

In Ada, it is possible to create a structure containing Boolean information according to a well defined representation with what is called the representation clause, and converting it into the ad hoc container with the unverified conversions functions that it took derive from the suitable generic package.

That's good, and used in many occasions in "Ada for Automation", but the author does not know whether it is effective to deal with such structures when the data is used in multiple Boolean equations, and it sounds complex for the automation that he remains.

"Ada for Automation" therefore provides the following conversion procedures:

```

package A4A.Library.Conversion is
  procedure Bytes_To_Word (LSB_Byte : in Byte;
                           MSB_Byte : in Byte;
                           Word_out  : out Word);

  procedure Word_To_Bytes (Word_in   : in Word;
                           LSB_Byte  : out Byte;
                           MSB_Byte  : out Byte);

  procedure Words_To_DWord (LSW_Word : in Word;
                             MSW_Word : in Word;
                             DWord_out : out DWord);

  procedure DWord_To_Words (DWord_in : in DWord;
                             LSW_Word : out Word;
                             MSW_Word : out Word);

  procedure Byte_To_Booleans (Byte_in : in Byte;
                               Boolean_out00 : out Boolean;
                               Boolean_out01 : out Boolean;
                               Boolean_out02 : out Boolean;
                               Boolean_out03 : out Boolean;
                               Boolean_out04 : out Boolean;
                               Boolean_out05 : out Boolean;
                               Boolean_out06 : out Boolean;
                               Boolean_out07 : out Boolean
                              );

  procedure Booleans_To_Byte (
    Boolean_in00 : in Boolean := false;
    Boolean_in01 : in Boolean := false;
    Boolean_in02 : in Boolean := false;
    Boolean_in03 : in Boolean := false;
    Boolean_in04 : in Boolean := false;
    Boolean_in05 : in Boolean := false;
    Boolean_in06 : in Boolean := false;
    Boolean_in07 : in Boolean := false;
    Byte_out    : out Byte
  );

  procedure Word_To_Booleans (Word_in : in Word;...);
  procedure Booleans_To_Word (...);
  procedure DWord_To_Booleans (DWord_in : in DWord;...);
  procedure Booleans_To_DWord (...);

end A4A.Library.Conversion;

```

12.3 Analog processing

The measure acquired, temperature, pressure, speed, etc. is provided in digital form, mostly an integer, a number of points representing the extent of the measure, which must be scaled, to get for example, a normalized value in the range [-100%, 100 %] or [0, 100%] or a scaled value in engineering units [-30, + 100 °C], [0, 50 Bars], [-10, 300 km/h] ...

12.3.1 Scale

So we find some scaling functions.

```

package A4A.Library.Analog is

  function Scale_In
    (X      : Integer;
     Xmin   : Integer;
     Xmax   : Integer;
     Ymin   : Float;
     Ymax   : Float)
    return Float;
  -- returns the value scaled :
  --  $Y = Ymax + ((X - Xmax) * (Ymax - Ymin) / (Xmax - Xmin))$ 

  function Scale_Out
    (X      : Float;
     Xmin   : Float;
     Xmax   : Float;
     Ymin   : Integer;
     Ymax   : Integer)
    return Integer;
  -- returns the value scaled :
  --  $Y = Ymax + ((X - Xmax) * (Ymax - Ymin) / (Xmax - Xmin))$ 

```

12.3.2 Limits

It is often interesting to limit the values that can take control variables.

Thus for example, on a hot-cold regulation, while the PID controller can produce normalized values in the range [-100%, +100%], we will send to the steam proportional valve a limited setpoint in interval [0%, +100%]. For the cold it is left as an exercise ...

```

function Limits
  (X      : Float;
   Ymin   : Float;
   Ymax   : Float)
  return Float;
  -- returns the value limited to the bounds

```

12.3.3 Ramp

Similarly, it may be desirable to limit the rate of change of the control variable which could otherwise have deleterious effects.

We may also need to generate a ramp from a step input.

The Ramp procedure, called in a periodic task performs this function. The gradient must be calculated based on the period of the task.

```

procedure Ramp
  (Value_In   : in Float;
   Gradient   : in Gradient_Type;
   Value_Out  : in out Float
  );
  -- has to be called in a periodic task
  -- the gradient has to be calculated accordingly

```

12.3.4 PID

This object implements the well-known algorithm, since available on Wikipedia, of the PID controller.

The cyclical function must of course be called in a periodic task and period must be entered.

The boolean "Initialize" resets to 0 the internal variables.

Instantiation

```
with A4A.Library.Analog.PID; use A4A.Library.Analog;

package A4A.User_Objects is

    My_PID_1          : PID.Instance;
    -- My PID Controller 1
```

Use

```
My_PID_1.Cyclic
(Set_Point           => My_PID_1_SP,
 Process_Value       => My_PID_1_PV,
 Kp                  => My_PID_1_Kp,
 Ki                  => My_PID_1_Ki,
 Kd                  => My_PID_1_Kd,
 Initialise          => My_PID_1_Init,
 Period_In_Milliseconds => 100,
 Manipulated_Value   => My_PID_1_MV
);
```

12.3.5 Thresholds

This object is a thresholds box that can be used for the control, signaling, alarm. It has a hysteresis and six adjustable thresholds.

Instantiation

```
with A4A.Library.Analog.Threshold; use A4A.Library.Analog;

package A4A.User_Objects is

    My_Thresholds_1 : Threshold.Instance;
    -- My Thresholds Box 1
```

Use The procedure "Initialize" allows to assign thresholds and hysteresis.

12.4 Timers

The timers are objects, we must first instantiate before using them.

12.4.1 TON

Delay on the rise.

Instantiation

```
with A4A.Library.Timers.TON; use A4A.Library.Timers;

package A4A.User_Objects is

    Tempo_TON_1      : TON.Instance;
    -- My Tempo TON 1
```

Use

```
with Ada.Real_Time;

    Elapsed_TON_1 : Ada.Real_Time.Time_Span;

    -- Could be simulate
    Tempo_TON_1.Cyclic (Start   => MyPump13Coil,
                       Preset  => Ada.Real_Time.Milliseconds (500),
                       Elapsed => Elapsed_TON_1,
                       Q       => MyPump13FeedBack);
```

12.4.2 TOFF

Release delay.

Instantiation

```
with A4A.Library.Timers.TOFF; use A4A.Library.Timers;

package A4A.User_Objects is

    Tempo_TOFF_1      : TOFF.Instance;
    -- My Tempo TOFF 1
```

Use

```
with Ada.Real_Time;

    Elapsed_TOFF_1 : Ada.Real_Time.Time_Span;

    -- TOFF little test
    Tempo_TOFF_1.Cyclic (Start   => Test_TOFF,
                       Preset  => Ada.Real_Time.Milliseconds (500),
                       Elapsed => Elapsed_TOFF_1,
                       Q       => Test_TOFF_Q);
```

12.4.3 TPULSE

Timing pulse.

Instantiation

```
with A4A.Library.Timers.TPULSE; use A4A.Library.Timers;

package A4A.User_Objects is

    Tempo_TPULSE_1      : TPULSE.Instance;
    -- My Tempo TPULSE 1
```

Use

```
with Ada.Real_Time;

    Elapsed_TPULSE_1 : Ada.Real_Time.Time_Span;

    -- TPULSE little test
    Tempo_TPULSE_1.Cyclic (Start   => Test_TPULSE,
                       Preset  => Ada.Real_Time.Milliseconds (500),
                       Elapsed => Elapsed_TOFF_1,
                       Q       => Test_TPULSE_Q);
```

12.5 Components

Components are objects that inherit from the class "Device", and that we must first instantiate before using them.

12.5.1 Device

Components represent physical entities of the system to automate. These are the pumps, valves, various and varied sensors, regulators, etc.

When the Process Man or mechanic has done its job, these components are identified by a function code and an identification number.

The component "Device" is the parent of all components with an identification.

This identification can then be used to track an alarm or a change of state, for example.

This is a virtual component which can not be instantiated as such.

12.5.2 Alarm Switch

This component is used to trigger an alarm if the alarm condition is satisfied.

- it is necessary firstly to filter false alarms, it is the role of the timeout,
- secondly memorize the alarm.

The input "Ack" is used to acknowledge the alarm.

The input "Inhibit" allows the inhibition of the alarm.

Instantiation

```
with A4A.Library.Devices.Alarm_Switch; use A4A.Library.Devices;

package A4A.User_Objects is

  LS12_AL          : Alarm_Switch.Instance :=
    Alarm_Switch.Create
      (Id           => "LS12",
       TON_Preset => 2000);
  -- Level Alarm Switch Instance
```

Use

```
LS12_AL.Cyclic(Alarm_Cond => not Level_Switch_12,
               Ack        => Ack_Faults,
               Inhibit    => False);

-- MyPump13 Command
Condition_Auto := not LS12_AL.is_Faulty and Valve_14.is_Open;
```

12.5.3 Contactor

This component represents a contactor or actuator controlled by this contactor.

- a delayed alarm is produced if the feedback does not confirm the operated state.

The input "Ack" is used to acknowledge the alarm.

Instantiation

```
with A4A.Library.Devices.Contactor; use A4A.Library.Devices;

package A4A.User_Objects is

  MyPump13      : Contactor.Instance :=
    Contactor.Create (Id => "Pump13");
  -- Pump Instance
```

Use

```
-- MyPump13 Command
Condition_Auto := not LS12_AL.is_Faulty and Valve_14.is_Open;

Condition_Manu := True;

MyPump13Cmd_On :=
  (Auto and Condition_Auto)
  or (Manu and Condition_Manu);

MyPump13.Cyclic (Feed_Back => MyPump13FeedBack,
                 Ack       => Ack_Faults,
                 Cmd_On    => MyPump13Cmd_On,
                 Coil      => MyPump13Coil);

-- Status
MyPump13IsOn := MyPump13.is_On;
```

12.5.4 Valve

This component is a normally closed valve.

- a delayed alarm is produced if the position limit switches do not confirm the operated state.

The input "Ack" is used to acknowledge the alarm.

Instantiation

```
with A4A.Library.Devices.Valve; use A4A.Library.Devices;

package A4A.User_Objects is

  Valve_14      : Valve.Instance :=
    Valve.Create
      (Id          => "XV14",
       TON_Preset => 5000); -- a slow valve
  -- Valve Instance
```

Use

```
Valve_14.Cyclic (Pos_Open   => Valve_14_Pos_Open,
                 Pos_Closed => Valve_14_Pos_Closed,
                 Ack        => Ack_Faults,
                 Cmd_Open   => Valve_14_Cmd_Open,
                 Coil       => Valve_14_Coil);

-- MyPump13 Command
Condition_Auto := not LS12_AL.is_Faulty and Valve_14.is_Open;
```

Chapter 13

Bibliography

13.1 Books

- [1] [barnes12] John Barnes. Programming in Ada 2012. Cambridge University Press. ISBN 978-1107424814.
 - [2] [barnes05] John Barnes. Programming in Ada 2005. Addison-Wesley. ISBN 0-32-134078-7.
 - [3] [burns-wellings] Alan Burns & Andy Wellings. Concurrent and Real-Time Programming in Ada. Cambridge University Press. ISBN 978-0-521-86697-2.
 - [4] [pert] McCormick, Singhoff & Hughes. Building Parallel, Embedded, and Real-Time Applications with Ada. Cambridge University Press. ISBN 978-0-521-19716-8.
-

Chapter 14

Glossary

PLC

Programmable Logic Controller.

Fieldbus

A means of communication between distributed devices.

Colophon

This book is composed in plain text in the format [Asciidoc](#) and processed to produce HTML or PDF files.
