

Take Off!



Ada for Automation

Freedom and Power for Control Engineers

Ada for Automation Demo Application

020 a4a-k1-cli

Stéphane LOS

Version 2022.05, 2022-05-31

Table of Contents

1. Description	1
1.1. Ada for Automation	1
1.2. This demo application	1
2. Projects diagram	2
3. License	3
4. Building	4
5. Running	5
6. Directories	6
7. Application	7
7.1. Deployment diagram	7
7.2. Activity diagram	8
7.3. Modbus TCP IO Scanning Configuration	11
7.4. Modbus TCP Server Configuration	13
7.5. User objects Definition	14
7.6. User Functions	15
7.7. User Application	18

Chapter 1. Description

1.1. Ada for Automation

[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 ModbusTCP client or server, or a Modbus RTU master or slave.

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.

With the help of [GtkAda](#), the binding to the [Graphic Tool Kit](#), one can design Graphical User Interfaces.

Thanks to [Gnoga](#), built on top of [Simple Components](#), it is also possible to provide a Web User Interface.

Nice addition is the binding to the [Snap7](#) library which allows to communicate with SIEMENS S7 PLCs using S7 Communication protocol ISO on TCP (RFC1006).

Of course, all the Ada ecosystem is available.

Using Ada bindings, C, C++, Fortran libraries can also be used.

And, since it is Ada, it can be compiled using the same code base to target all major platforms.

1.2. This demo application

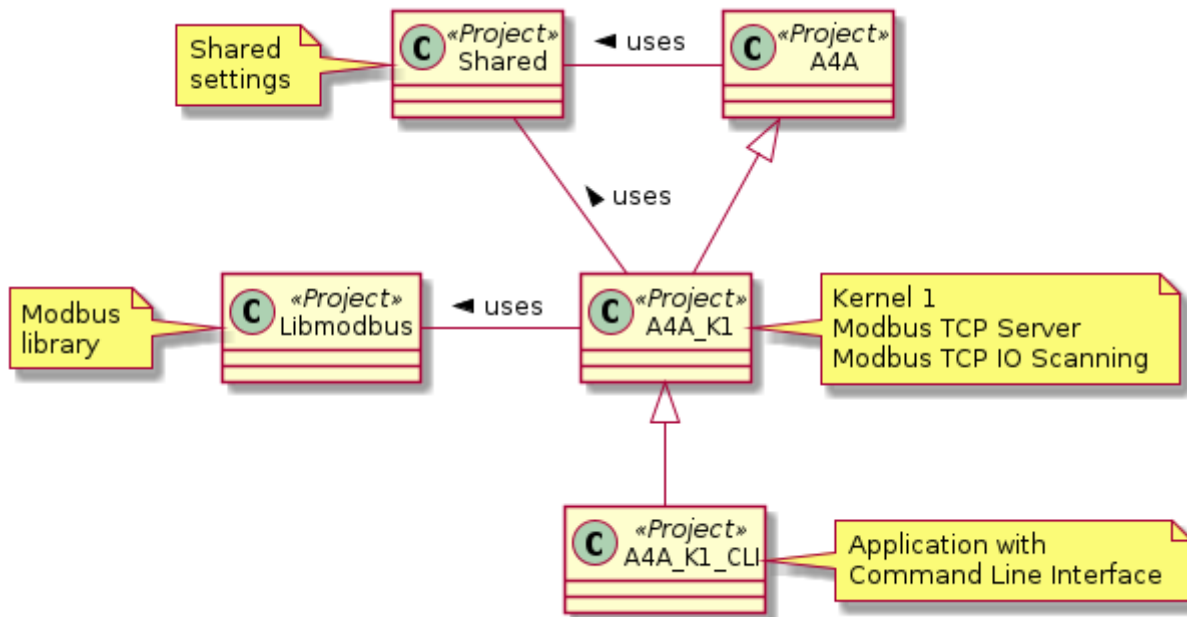
This is a demo application featuring:

- a basic command line interface,
- a kernel with a Modbus TCP Server and Modbus TCP IO Scanning (K1),
- a trivial application that plays with 16 push buttons and 16 LEDs.

This application is meant to play with **010 a4a_piano**.

Chapter 2. Projects diagram

The following picture shows the diagram of projects :



Chapter 3. License

Those files are included in the **Ada for Automation** root folder :

COPYING3

The GPL License you should read carefully. GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

COPYING.RUNTIME

GCC RUNTIME LIBRARY EXCEPTION Version 3.1, 31 March 2009

Hence, each source file contains the following header :

```
-----  
--                               Ada for Automation                               --  
--                               --  
--                               Copyright (C) 2012-2023, Stephane LOS          --  
--                               --  
-- This library is free software; you can redistribute it and/or modify it --  
-- under terms of the GNU General Public License as published by the Free --  
-- Software Foundation; either version 3, 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 MERCHAN- --  
-- TABILITY or FITNESS FOR A PARTICULAR PURPOSE.                               --  
--                               --  
-- As a special exception under Section 7 of GPL version 3, you are granted --  
-- additional permissions described in the GCC Runtime Library Exception, --  
-- version 3.1, as published by the Free Software Foundation.                --  
--                               --  
-- You should have received a copy of the GNU General Public License and --  
-- a copy of the GCC Runtime Library Exception along with this program; --  
-- see the files COPYING3 and COPYING.RUNTIME respectively. If not, see --  
-- <http://www.gnu.org/licenses/>. --  
--                               --  
-----
```

Chapter 4. Building

The provided makefile uses [GPRbuild](#) and provides six targets:

- all : builds the executable,
- app_doc : creates the documentation of the source code,
- clean : cleans the space.

Additionally one can generate some documentation using [Asciidoctor](#) with :

- read_me_html : generates the README in HTML format,
- read_me_pdf : generates the README in PDF format,
- read_me : generates the README in both formats.

Chapter 5. Running

This application is meant to play with **010 a4a_piano**.

Of course, it can also play with a physical device of your own.

In a console:

Build the application:

```
make
```

Optionally create the documentation:

```
make app_doc
```

Run the application:

```
make run
```

Use Ctrl+C to exit.

Optionally clean all:

```
make clean
```

Chapter 6. Directories

bin

Where you will find the executable.

doc

The place where [GNATdoc](#) would create the documentation.

obj

Build artifacts go here.

src

Application source files.

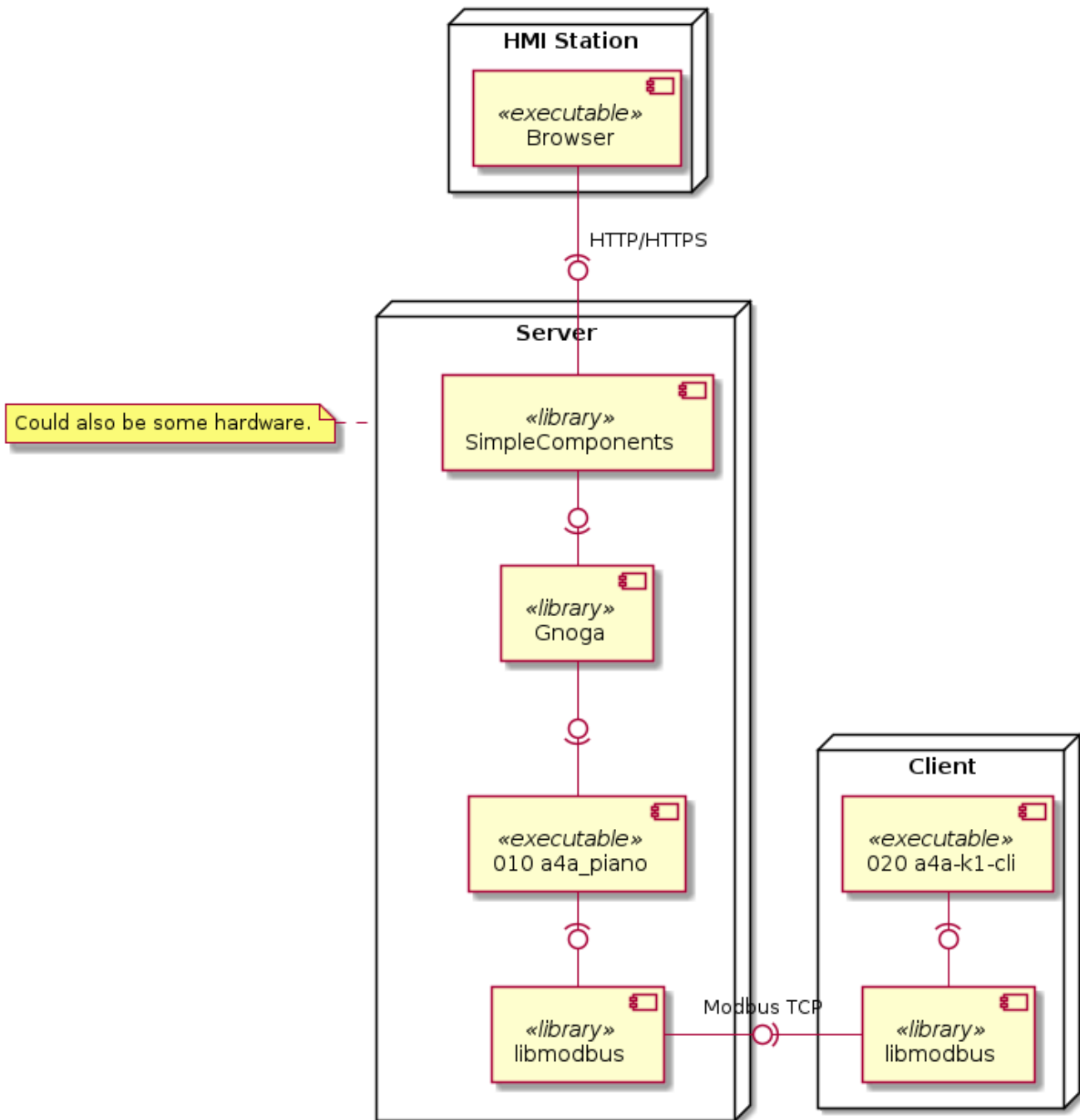
Chapter 7. Application

This is a basic **Ada for Automation** application which implements Modbus TCP IO Scanning and plays with 16 push buttons and 16 LEDs.

It has a Command Line User Interface and sends Modbus TCP requests to the server reading Input bits and writing Coils.

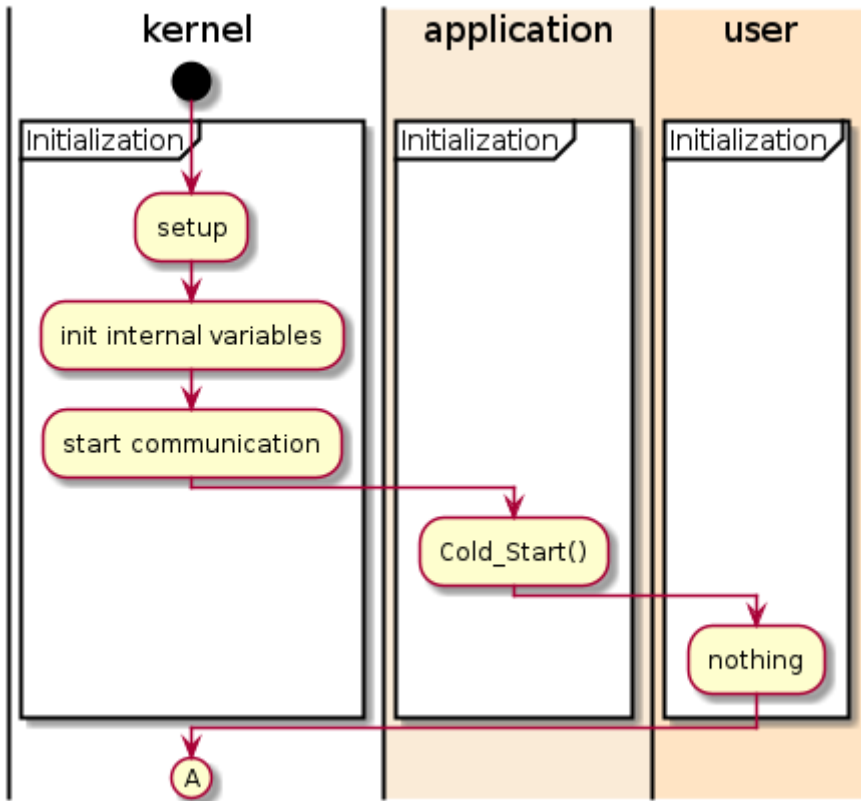
Modbus TCP Server is not used at the moment.

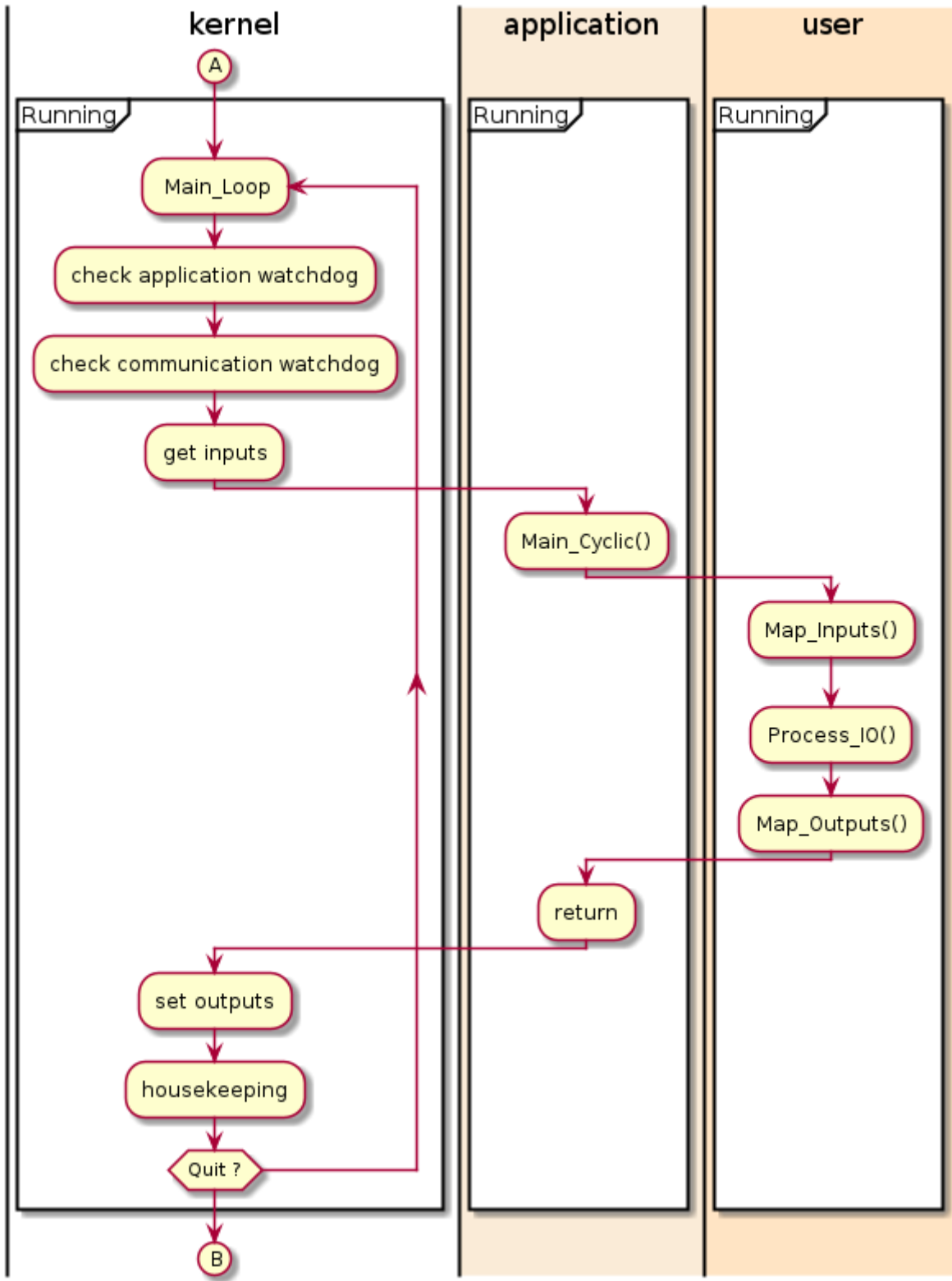
7.1. Deployment diagram

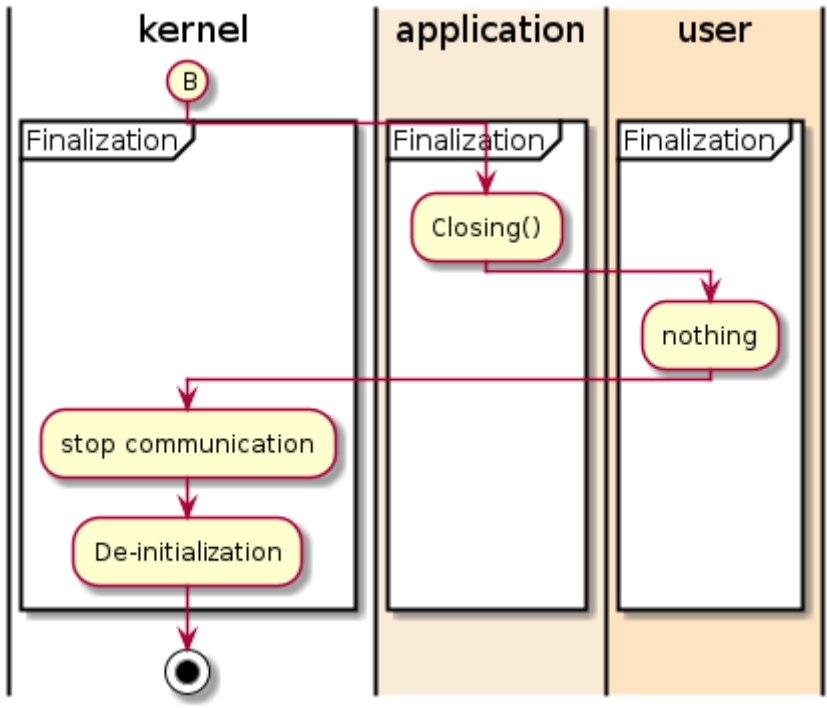


7.2. Activity diagram

The Kernel manages the communication channels and provides an interface to those, namely the packages "A4A.Memory.MBTCP_IOScan" and "A4A.Memory.MBTCP_IOSlave".







7.3. Modbus TCP IO Scanning Configuration

For each Modbus TCP Server define one client configuration task and declare all clients configurations in the array

"../src/a4a-application-mbtcp_clients_config.ads"

```
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   => 2, -- ③
   Enabled          => True,
   Debug_On        => False,
   Task_Period_MS   => 10,
   Retries          => 3,
   Timeout          => 0.2,

   -- Using an IP Address
   -- Node    => To_Bounded_String ("127.0.0.1"), -- ①
   -- or a Node name, i.e. example.com
   -- Node    => To_Bounded_String ("localhost"), -- ①
   -- This one for the Docker demo
   Node      => To_Bounded_String ("a4a_piano"), -- ①
   Service   => To_Bounded_String ("1504"), -- ②
   -- Standard : 502

   Commands => -- ④
   (
     --
     --
     -- Action Enabled Multiple Shift Number Remote Local
     1 =>
       (Read_Input_Bits,      True,    10,    0,    16,    0,    0),
     2 =>
       (Write_Bits,          True,    10,    5,    16,    0,    0)
   )
  );

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

```

Node    => To_Bounded_String ("127.0.0.1"),
Service => To_Bounded_String ("1503"), -- My own MBTCP server

Commands =>
(
  --
  --
  --          Period          Offset Offset
  --          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 configurations in the array
-- The kernel will create those clients accordingly

MBTCP_Clients_Configuration : Client_Configuration_Access_Array :=
  (1 => Config1'Access, -- ⑤
   2 => Config2'Access);

end A4A.Application.MBTCP_Clients_Config;

```

- ① Modbus TCP Server IP Address : 127.0.0.1 (localhost)
- ② Modbus TCP Server port : 1504 (A4A_Piano)
- ③ Commands number : 2 (since we declare two commands in the array)
- ④ Commands array : one read and one write commands
- ⑤ Configurations array : add our configuration

7.4. Modbus TCP Server Configuration

"../src/a4a-application-mbtcp_server_config.ads"

```
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     => 1503); -- ①

end A4A.Application.MBTCP_Server_Config;
```

① Modbus TCP Server port : 1503 (default)

7.5. User objects Definition

"/src/a4a-user_objects.ads"

```
package A4A.User_Objects is
-----
-- User Objects creation
-----

First_Cycle : Boolean := True;
Output_Byte : Byte := 0; -- ③

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

TON_1_Q      : Boolean := False;

Cmd_Byte     : Byte := 0; -- ①
Pattern_Byte : Byte := 0; -- ②

end A4A.User_Objects;
```

- ① 8 Input bits are read that form the Command byte, which is also output in 8 Coils.
- ② 8 Input bits are read that form the Pattern byte.
- ③ 8 Coils are written that reflect the Output byte.

7.6. User Functions

"/src/a4a-user_functions.adb"

```
package body A4A.User_Functions is

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

procedure Map_Inputs is -- ①
begin

    Booleans_To_Byte (Boolean_in00 => MBTCP_IOScan.Bool_Inputs (0),
                      Boolean_in01 => MBTCP_IOScan.Bool_Inputs (1),
                      Boolean_in02 => MBTCP_IOScan.Bool_Inputs (2),
                      Boolean_in03 => MBTCP_IOScan.Bool_Inputs (3),
                      Boolean_in04 => MBTCP_IOScan.Bool_Inputs (4),
                      Boolean_in05 => MBTCP_IOScan.Bool_Inputs (5),
                      Boolean_in06 => MBTCP_IOScan.Bool_Inputs (6),
                      Boolean_in07 => MBTCP_IOScan.Bool_Inputs (7),
                      Byte_out      => Cmd_Byte);

    Booleans_To_Byte (Boolean_in00 => MBTCP_IOScan.Bool_Inputs (8),
                      Boolean_in01 => MBTCP_IOScan.Bool_Inputs (9),
                      Boolean_in02 => MBTCP_IOScan.Bool_Inputs (10),
                      Boolean_in03 => MBTCP_IOScan.Bool_Inputs (11),
                      Boolean_in04 => MBTCP_IOScan.Bool_Inputs (12),
                      Boolean_in05 => MBTCP_IOScan.Bool_Inputs (13),
                      Boolean_in06 => MBTCP_IOScan.Bool_Inputs (14),
                      Boolean_in07 => MBTCP_IOScan.Bool_Inputs (15),
                      Byte_out      => Pattern_Byte);

end Map_Inputs;

procedure Map_Outputs is -- ②
begin

    Byte_To_Booleans (Byte_in      => Cmd_Byte,
                     Boolean_out00 => MBTCP_IOScan.Bool_Outputs (0),
                     Boolean_out01 => MBTCP_IOScan.Bool_Outputs (1),
                     Boolean_out02 => MBTCP_IOScan.Bool_Outputs (2),
                     Boolean_out03 => MBTCP_IOScan.Bool_Outputs (3),
                     Boolean_out04 => MBTCP_IOScan.Bool_Outputs (4),
                     Boolean_out05 => MBTCP_IOScan.Bool_Outputs (5),
                     Boolean_out06 => MBTCP_IOScan.Bool_Outputs (6),
                     Boolean_out07 => MBTCP_IOScan.Bool_Outputs (7));

    Byte_To_Booleans (Byte_in      => Output_Byte,
                     Boolean_out00 => MBTCP_IOScan.Bool_Outputs (8),
                     Boolean_out01 => MBTCP_IOScan.Bool_Outputs (9),
```

```

        Boolean_out02 => MBTCP_IOScan.Bool_Outputs (10),
        Boolean_out03 => MBTCP_IOScan.Bool_Outputs (11),
        Boolean_out04 => MBTCP_IOScan.Bool_Outputs (12),
        Boolean_out05 => MBTCP_IOScan.Bool_Outputs (13),
        Boolean_out06 => MBTCP_IOScan.Bool_Outputs (14),
        Boolean_out07 => MBTCP_IOScan.Bool_Outputs (15));

end Map_Outputs;

procedure Map_HMI_Inputs is
begin

    null;

end Map_HMI_Inputs;

procedure Map_HMI_Outputs is
begin

    null;

end Map_HMI_Outputs;

procedure Process_I0 is -- ③

    Elapsed_TON_1 : Ada.Real_Time.Time_Span;

begin

    if First_Cycle then

        Output_Byte := Pattern_Byte;

        First_Cycle := False;

    end if;

    Tempo_TON_1.Cyclic (Start => not TON_1_Q,
                        Preset => Ada.Real_Time.Milliseconds (500),
                        Elapsed => Elapsed_TON_1,
                        Q      => TON_1_Q);

    if TON_1_Q then

        case Cmd_Byte is

            when 0 =>
                Output_Byte := ROR (Value => Output_Byte, Amount => 1);

            when 1 =>
                Output_Byte := ROL (Value => Output_Byte, Amount => 1);
        end case;
    end if;
end Process_I0;

```

```
        when others => Output_Byte := Pattern_Byte;

        end case;

    end if;

end Process_IO;

end A4A.User_Functions;
```

User functions are defined to :

- ① get the inputs from the IO Scanner,
- ② set IO Scanner outputs,
- ③ process the data.

7.7. User Application

"/src/a4a-application.adb"

```
package body A4A.Application is

  procedure Cold_Start is
  begin

    null;

  end Cold_Start;

  procedure Closing is
  begin

    null;

  end Closing;

  procedure Main_Cyclic is
    My_Ident : constant String := "A4A.Application.Main_Cyclic";
  begin

    Map_Inputs; -- ①

    Map_HMI_Inputs;

    -- Playing with tasks interface
    Main_Outputs.X := Main_Inputs.A;
    Main_Outputs.Y := Main_Inputs.B;
    Main_Outputs.Z := Main_Inputs.C;

    Process_IO; -- ②

    Map_Outputs; -- ③

    Map_HMI_Outputs;

  exception

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

      Program_Fault_Flag := True;

  end Main_Cyclic;

  procedure Periodic1_Cyclic is
    My_Ident : constant String := "A4A.Application.Periodic1_Cyclic";
```

```
begin
```

```
-- Playing with tasks interface  
Periodic1_Outputs.A := not Periodic1_Inputs.X;  
Periodic1_Outputs.B := Periodic1_Inputs.Y + 2;  
Periodic1_Outputs.C := Periodic1_Inputs.Z + 1;
```

```
exception
```

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

```
    Program_Fault_Flag := True;
```

```
end Periodic1_Cyclic;
```

```
function Program_Fault return Boolean is
```

```
begin
```

```
    return Program_Fault_Flag;
```

```
end Program_Fault;
```

```
end A4A.Application;
```

The application cyclically :

- ① gets the inputs from the IO Scanner,
- ② processes the data,
- ③ sets IO Scanner outputs.