

# BASIC MANUAL TO UNDERSTAND THE SOULISS LIBRARIES

Souliss Main Group (english) <https://groups.google.com/forum/#!forum/souliss>  
Souliss Italian Team (Italian) <https://groups.google.com/forum/#!forum/souliss-it>  
Souliss Spanish team (Spanish) <https://groups.google.com/forum/#!forum/souliss-es>  
Github of the project <https://github.com/souliss>

Thanks to all the people who make that project possible.

## Table of Contents

SOULISS RELEASES (UNTIL 21MAY15):.....	3
INTRODUCTION:.....	8
<a href="#">An example configuration</a> .....	8
<a href="#">List of available configuration files</a> .....	9
<a href="#">List of available board configuration files</a> .....	10
SOULISS.H.....	11
TYPICAL.H.....	15
GETCONFIG.H.....	23
SPEAKEASY.H.....	25
T1N.CPP.....	35
T2N.CPP.....	65

**uint8\_t** – tipo de dato de 8 bits sin signo

**int8\_t** – tipo de dato de 8 bits con signo

**#include “libreria.h”** - busca el fichero en el ambito de trabajo y a continuación en el ambito estandard

**#include <libreria.h>** - busca el fichero únicamente en el ámbito estandard

## **SOULISS RELEASES (UNTIL 21MAY15):**

Souliss

Copyright (C) 2011-2015

Release note for **revision v7.0.3, on 07/05/2015**

Revised configuration structure and dynamic addressing features, extended support for battery operated devices.

- Configuration is only available through inSketch or changing the configuration files
- Quick Configuration is deprecated.
- Dynamic Addressing works also in case of complex network
- EEPROM is used to retain network configuration for either Gateway and Peers
- Review of all examples
- Zeroconfig examples
- Introduction of Publish/Subscribe user methods
- Massive commands can be used between nodes
- Media 3 has been moved from MAC broadcast to IP broadcast
- General bugfix

Release note for revision Alpha 6.1.4, on 25/01/2015

Minor bugfix, last release code for branch Alpha 6.

Release note for revision Alpha 6.1.3, on 18/12/2014

General improvement and minor bugfix

- Added software filtering for digital inputs
- Added support for T1A Digital Input
- Rollback autoaddressing code to Alpha 5.2
- Added support for "IoTuino Radio Only" using an external Ethernet transceiver
- nRF24L01 CS and EN pins are moved to inSketch
- Fix on SPI CS for Atmega32U4
- Bugfix for case sensitive on DHCP code
- Bugfix on T11 and DigInHold
- Other minor bugfix

Thanks to: Juan Luis, Juan Pinto, Fulvio Spelta,  
Marco Pozzuolo, Saverio Sbrana

Release note for revision Alpha 6.1.2, on 07/11/2014

Modified T22 logic and general bugfixes

Release note for revision Alpha 6.1.1, on 13/10/2014

Support for Moteino boards and fix the Alpha 6.0.3 this release include preliminary functionality that needs to be consolidated in future.

- Introduction of Action Messages (preliminary)
- Massive commands can be used between nodes (preliminary)
- Preliminary support for Moteino and HopeRF RFM69CW/RFM69HCW
- Media 3 has been moved from MAC broadcast to IP broadcast (preliminary)
- Iotuino USART drivers are now based on SerialPort by William Greiman
- Improvement on broadcasting
- New typical retrieval approach bugfixed

Release note for revision Alpha 6.0.3, on 23/09/2014

Fix a major bug while retrieving typicals for logics loaded in the network nodes

Release note for revision Alpha 6.0.2, on 14/09/2014

Major improvement on USART drivers for vNet

- vNet USART driver buffer handling improvement
- vNet USART max frame is now a parameter
- vNet USART node startup delay is introduced
- vNet USART retry in case of bus busy
- MaCaco TYP commands are now in broadcast (before unicast was used)

Release note for revision Alpha 6.0.1, on 22/08/2014

Modbus TCP/RTU interface is back available. Introduced a buffer for the XML interface for either TCP and UDP cases to handle multiple commands in short time.

- Modbus TCP/RTU interface
- Modified the memory handling, save more RAM on Peers
- Nodes can dynamically join a Gateway even if has a fixed vNet address
- Support for Multicast
- Automatic update of the routing table in case of dynamic join of a gateway
- vNet Autoreset
- Command buffer for HTTP/XML and UDP/XML Interface (bugfix by Alexander Mathis)
- The openHAB Interfaces are now generic and called XMLServer Interfaces
- All Souliss DigIn can now returns the results without affect slots values
- Support of DINo over RS485
- Examples with Modbus TCP over DINo with plug&play
- Support for HF-LPT200 (preliminary)
- Bugfix

Release note for revision Alpha 5.2, on 22/05/2014

A new XML data interface over UDP has been introduced, based on LASTIN allows event driven data retrieving in ASCII/XML format. Support for nRF24L01 and nRF24L01+ 2.4 GHz radio and Wiznet W5500 Ethernet controller.

- |   |                    |
|---|--------------------|
| - Extended openHAB HTTP/XML interface on uIP    | - Juan Pinto       |
| - Completed LASTIN mode                         | - Fulvio Spelta    |
| - UDP/XML Interface (Event based communication) | - Fulvio Spelta    |
| - Battery friendly subscription mode introduced | - Philippe Peltier |
| - vNet drivers for Wiznet W5500                 | -                  |
| - vNet drivers for nRF24L01 and nRF24L01+       | -                  |
| - New examples                                  | -                  |

Flavio Piccolo

- Bugfix

Release note for revision Alpha 5.1, on 09/03/2014

The introduction of PERSISTENCE and LASTIN modes makes a gateway node able to provide data to external sources using polling protocols.

External protocols are now referred as Interfaces and are available either on gateway as peer nodes.

- Introduced openHAB interface based on HTTP/XML (thanks to Fulvio Spelta)
- The INTERFACES takes place of HTTP and Modbus GATEWAY
- The INTERFACES is now handled directly on MaCaco and has PERSISTENCE and LASTIN modes available
- Bugfix for USART over RS485 (thanks to Marco Pozzuolo and Juan Pinto)

Release note for revision Alpha 5.0.4, on 12/01/2014

Introduction of USART drivers in Quick Configuration and preliminary support for Arduino Ethernet Library methods, this ensure compatibility with third party libraries and doesn't affect Souliss/vNet communication.

Release note for revision Alpha 5.0.3, on 18/12/2013

Support for DINo v2 completed with addition of Board Configuration files contained in bconf/ folder, that include automatically all the Quick Configuration parameters for that sketch, without need of user intervention of QuickConf.h file.

- Board Configuration files for inSketch mode added
- KMP Electronics DINo v2 full support completed
- Bugfix for IR-RGB Light, thanks to Juan Pinto
- Bugfix for Inputs method, thanks to Juan Luis Navarro.

Release note for revision Alpha 5.0.2, on 01/12/2013

Introduction of inSketch mode, this move selected configuration parameters from the configuration file to the sketch. Removed legacy non-pass-through mode, from this time the gateway cannot longer serve polling protocols.

- Introduced inSketch mode,
- Modbus protocol deprecated,
- Bugfix for DINo v2 support (still work in progress).

Release note for revision Alpha 5.0.1, on 13/11/2013

Introduced a revised memory handling for MaCaco in order reduce the RAM waste in case of non-gateway nodes, broadcasting features for configuration-less nodes

- Configuration-less network for simple architectures added
- Nodes can join a Souliss network in runtime
- Broadcast support for vNet
- PassThrough mode for MaCaco communication with User Interfaces
- Support for KMP Electronics DINo II relay board
- Added Wiznet W5200 drivers (Ethernet controller)
- HTTP/JSON running on nodes is deprecated
- Revised memory structure for MaCaco, save RAM on non-gateway nodes
- Bandwidth friendly approach for MaCaco retries mechanism
- Bug-fix for uIP/vNet IP source port handling (thanks to Antonino Fazio)
- Arduino IDE 1.0.x and 1.5.x supported as main releases

Release note for revision Alpha 4.5.1, on 10/09/2013

Introduced a macro based easy coding feature referred as Speak Easy

- Introduced Speak Easy
- Introduced an USART vNet Driver for RS485 communication (preliminary)
- Native support for DFRobots XBoard
- Support for ENC28J60 + AT86RF230 SuperNodes (thanks to Antonino Fazio)
- Bug-fix for interrupt handler of chibiduino stack
- Bug-fix on ENC28J60 vNet Driver

Release note for revision Alpha 4.5, on 18/07/2013

Review of analog values management and relevant logics (typicals), support for XBoard Relay from DFRobots and DHT temperature and humidity sensors

- Introduced half-precision floating and methods
- Native support for DHT sensors
- Native support for DFRobots XBoard Relay
- Bug-fix as per chibiduino stack v.1.00a

- General bug-fix for typicals (T16, T42)

Release note for revision Alpha 4.4.2, on 04/06/2013

Bugfix in vNet over IP drivers, now source port is handled properly also in case of source port masquering by NAT (thanks to Juan Luis).

Release note for revision Alpha 4.4.1, on 13/05/2013

Included updated documentation for

- MaCaco Light Data Protocol "User and Developers Guide"
- vNet Network Virtualization Layer "User Guide"
- Wireshark capture for MaCaco over vNet/UDP-IP

Release note for revision Alpha 4.4, on 07/05/2013

Extended support of Olimex boards

- Support for Olimex MOD-RGB
- Support for Olimex MOD-IO
- Modbus RemoteInputs fix (thanks to Giorgio Zoppetti)

Release note for revision Alpha 4.3, on 12/04/2013

Support for AirQ Networks wireless 433 MHz boards (needs download of Air sNET library)

- Support for AirQ Shield
- Support for AirQ 305
- Bug fix.

Release note for revision Alpha 4.2, on 25/03/2013

Support for Open Electronics boards and improvement on multicast commands.

- Support for Open Electronics RGB Shield
- Support for Open Electronics ENC28J60 Ethernet Shield
- Reduced number of frame sent for multicast commands
- Common commands tuned for Modbus coil use.

Release note for revision Alpha 4.1, on 08/02/2013

Support Microchip MRF24WB0MA WiFi module and Olimex boards, introduction of Quick Configuration mode.

- Support for Olimex AVR-T32U4
- Support for Olimex OLIMEXINO-32U4
- Support for Olimex OLIMEXINO-328
- Support for Olimex MOD-ENC28J60
- Support for Olimex MOD-WIFI
- Support for Olimex MOD-IO2
- Support for Microchip MRF24WB0MA (up to firmware relase 0x1207)
- Quick Configuration mode

Release note for revision Alpha 4, on 22/12/2012

Support for Modbus and extended MaCaco for user interfaces, all communication based on binary protocol to increase the performances. Improvement on RAM handling.

- Introduction of oFrame for data buffer handling,

- Extended MaCaco for user interfaces,
- Use of UDP for vNet based communication instead of TCP,
- Modbus RTU and TCP Slave,
- HTTP Server for remote commands,
- New Server/Client TCP APIs,
- New Examples,
- Bug fix.

Release note for revision Alpha 3.1, on 30/07/2012

Bugfix for ENC28J60 and first release of plinio drivers

- Binary FSK using ATmega328P (preliminary),
- New JSON Server with support for one and two commands at same time,
- Bug fix,
- Enhanced examples.

Release note for revision Alpha 3, on 18/07/2012

Support for the Android Client application, integration with anti-theft system and tools for communication debugging:

- Support for ENC28J60,
- Introduction of Raw Ethernet communication,
- Watchdog chain and typicals for anti-theft integration,
- Debug mode for vNet and MaCaco using serial port,
- Enhanced scheduler and examples,
- Friendly names for Souliss examples,
- Arduino IDE 1.

Release note for revision Alpha 2.3, on 11/04/2012

Support for the Android Client application:

- Integration with dedicated Android Cleint adding a new JSON array structure,

Release note for revision Alpha 2.2, on 02/02/2012

Bugfix

Release note for revision Alpha 2.1, on 01/02/2012

Support for nodes dedicated as JSON Server.

Release note for revision Alpha 2, on 07/01/2012

Support for user customizable interfaces is added, following additional features are provided:

- Compatibility with standard Ethernet class (2 sockets),
- JSON Server for User Interfaces,
- Browser based User Interface,
- Support for new typicals (ready to use logics for Home Automation).

Release note for revision Alpha, on 23/11/2011

An smart home framework with Ethernet and 2.4 Ghz Point to Point Wireless support.

Supported boards are:

- Freaklabs Chibiduino (2.4 GHz Wireless)
- Arduino 2009/UNO with Ethernet Shield (Wiznet W5100),
- Arduino Ethernet.

Development environment Arduino IDE 0022.

## INTRODUCTION:

The framework configuration is just a collection of settings required to specify the architecture, transceiver and I/O module used in your node, in this way the proper code is automatically compiled and you can run the same sketch over different architectures.

The configuration is done through `#define` statements that are listed into the general configuration files that are located in `/conf` and `/bconf` folders, you can `#define` most of these setting into the sketch having the opportunity to bind sketch and relevant configuration together, this is called [INSKETCH](#).

## An example configuration

Is easier see how a sketch is configured than explain it, consider the [e01\\_HelloWorld](#), this is configured to run over an Arduino Ethernet or Arduino UNO/Leonardo/Mega with Ethernet shield. The configuration looks like the followings:

```
// Configure the framework
#include "bconf/StandardArduino.h"           // Use a standard Arduino
#include "conf/ethW5100.h"                     // Ethernet through Wiznet W5100
#include "conf/Gateway.h"                      // The main node is the Gateway, we
have just one node
#include "conf/Webhook.h"                      // Enable DHCP and DNS

// Include framework code and libraries
#include <SPI.h>
#include "Souliss.h"
```

The `#include "bconf/StandardArduino.h"` specify the board type (`bconf` stands for *board configuration*) as Standard Arduino, with a Wiznet W5100 Ethernet Shield (or embed on board like in Arduino Ethernet) `#include "conf/ethW5100.h"` and configured as a [Gateway](#).

The files **must be included before the `#include "Souliss.h"` statement, otherwise will not have any effect.**

## List of available configuration files

The following files are located in the /conf folder and are used to specify a single piece of functionality required

<b>File Name</b>	<b>Description</b>
DynamicAddressing.h	Enable the dynamic addressing
ethENC28J60.h	Use the Ethernet transceiver Microchip ENC28J60
ethW5100.h	Use the Ethernet transceiver Wiznet W5100
ethW5200.h	Use the Ethernet transceiver Wiznet W5200
ethW5500.h	Use the Ethernet transceiver Wiznet W5500
Gateway.h	Set the node as <a href="#">Gateway</a>
Gateway_wLastin.h	Set the node as <a href="#">Gateway</a> with Last In persistence
Gateway_wPersistence.h	Set the node as <a href="#">Gateway</a> with full persistence
HTTPInterface.h	Enable HTTP Interface
ModbusRTU.h	Enable Modbus RTU Interface
ModbusTCP.h	Enable Modbus TCP Interface
nRF24L01.h	Use the nRF24L01(+) 2.4 GHz radio
Sleep.h	Enable the Sleep functionality for battery operated devices
SmallNetwork.h	Reduce the network size and the relevant RAM usage
SuperNode.h	Set the node as SuperNode
uart.h	Use the USART to communicate with other nodes (RS485 and wireless-usart)
Webhook.h	Include DNS and DHCP, for Wiznet based devices
wifiMRF24.h	Use the Microchip MRF24 WiFi transceiver
WirelessExtender.h	Set the node as wireless extender
XMLInterface.h	Enable the XML Interface

You should refer to the guide of each of this functionality to understand when and how those can be used.

## List of available board configuration files

The following files are located in the /bconf folder and are used to specify a full board including all available functionality.

File Name	Description
Chibiduino_v1.h	Chibiduino, Arduino Duemilanove (compatible) with integrated 2.4 GHz radio
DevDuino_v2.h	Arduino UNO (compatible) with nRF24L01 socket for battery operated nodes
DINo_v1.h	Arduino Duemilanove (compatible) with relay and Microchip ENC28J60 Ethernet transceiver
DINo_v2.h	Arduino Leonardo (compatible) with relay, Wiznet W5200 Ethernet transceiver
DINo_v2_EthernetBridge_RS485.h	Arduino Leonardo (compatible) with relay, Wiznet W5200 Ethernet transceiver and RS485
DINo_v2_RS485.h	Arduino Leonardo (compatible) with relay and RS485
Moteino.h	Arduino compatible with MRF69 900 MHz radio
OlimexAVRT32U4.h	Arduino Leonardo (compatible) with special pinout and UEXT
Olimexino328.h	Arduino Duemilanove (compatible) with special pinout and UEXT
Olimexino32U4.h	Arduino Leonardo (compatible) with special pinout and UEXT
OlimexMOD-ENC28J60.h	Microchip ENC28J60 Ethernet transceiver for UEXT
OlimexMOD-IO.h	I/O Board for UEXT
OlimexMOD-IO2.h	I/O Board for UEXT
OlimexMOD-RGB.h	LED Board for UEXT
OlimexMOD-WIFI.h	Microchip MRF24 WiFi transceiver for UEXT
StandardArduino.h	Arduino board (Duemilanove, UNO, Leonardo or Mega)
XBoard.h	Arduino UNO (compatible)
XBoardRelay.h	Arduino Ethernet (compatible) with relay

## SOULISS.H

---

Souliss  
Copyright (C) 2013 Veseo

This program 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 3 of the License, or (at your option) any later version.

This program 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 program. If not, see <<http://www.gnu.org/licenses/>>.

Originally developed by Dario Di Maio

---

```
/*!
 \file
 \ingroup
 */
#ifndef SOULISSSKETCHMODE_H
#define SOULISSSKETCHMODE_H
```

---

```
/*!
 */
/*!
```

All code is compiled into the sketch, without creating middle object files. This let use #define that are globally recognized in the whole code, including sub-libraries.

```
*/
#include <Arduino.h>
#include "tools/types.h"
#include "Typicals.h"
#include "GetConfig.h" // need : ethUsrCfg.h, vNetCfg.h, SoulissCfg.h,
MaCacoCfg.h
```

```
#include "frame/MaCaco/MaCaco.h"
#include "frame/vNet/vNet.h"
```

```
#if defined(__AVR_ATmega1280__)
#    define MAXINPIN      69          // Max number of input pins
#elif defined(__AVR_ATmega1284P__)
#    define MAXINPIN      40          // Max number of input pins
#elif defined(__AVR_ATmega32U4__)
#    define MAXINPIN      29          // Max number of input pins
#else
```

```
#define MAXINPIN      29          // Max number of input pins
#endif

void Souliss_SetAddress(U16 addr, U16 subnetmask, U16 mysupernode);
void Souliss_SetLocalAddress(U8 *memory_map, U16 addr);
void Souliss_SetRemoteAddress(U8 *memory_map, U16 addr, U8 node);
U8 Souliss_GetTypicals(U8 *memory_map);
U8 Souliss_CommunicationChannel(U16 addr, U8 *memory_map, U8 input_slot, U8 output_slot,
U8 numof_slot, U8 subscr_chnl);
U8 Souliss_CommunicationChannels(U8 *memory_map);
void Souliss_BatteryChannels(U8 *memory_map, U16 addr);
U8 Souliss_HardcodedCommunicationChannel(U16 gateway_addr);
void Souliss_JoinNetwork();
void Souliss_SetIPAddress(U8* ip_address, U8* subnet_mask, U8* ip_gateway);
void Souliss_SetAddressingServer(U8 *memory_map);
void Souliss_SetDynamicAddressing();
U8 Souliss_DynamicAddressing (U8 *memory_map, const char id[], U8 size);
U8 Souliss_RemoteInput(U16 addr, U8 slot, U8 command);
U8 Souliss_RemoteInputs(U16 addr, U8 firstslot, U8 *commands, U8 numberof);
U8 Souliss_MassiveCommand(U16 addr, U8 typ, U8 command);
U8 Souliss_BroadcastMassiveCommand(U8 typ, U8 command);
U8 Souliss_Publish(U8 *memory_map, U16 message, U8 action);
U8 Souliss_MulticastPublish(U16 multicast_addr, U8 *memory_map, U16 message, U8 action);
U8 Souliss_PublishData(U8 *memory_map, U16 message, U8 action, U8* data, U8 message_len);
U8 Souliss_MulticastPublishData(U16 multicast_addr, U8 *memory_map, U16 message, U8
action, U8* data, U8 message_len);
U8 Souliss_Subscribe(U8 *memory_map, U16 message, U8 action);
U8 Souliss_SubscribeData(U8 *memory_map, U16 message, U8 action, U8* data, U8* len);
U8 Souliss_CommunicationData(U8 *memory_map, U8 *trigger);
U8 Souliss_Watchdog(U8 *memory_map, U16 chain_address, U8 chain_slot, U8
alarm_command);
U8 Souliss_DigIn(U8 pin, U8 value, U8 *memory_map, U8 slot, bool filteractive);
U8 Souliss_LowDigIn(U8 pin, U8 value, U8 *memory_map, U8 slot, bool filteractive);
U8 Souliss_DigIn2State(U8 pin, U8 value_state_on, U8 value_state_off, U8 *memory_map, U8
slot);
U8 Souliss_AnalogIn2Buttons(U8 pin, U8 value_button1, U8 value_button2, U8 *memory_map,
U8 slot);
U8 Souliss_LowDigIn2State(U8 pin, U8 value_state_on, U8 value_state_off, U8 *memory_map,
U8 slot);
U8 Souliss_DigInHold(U8 pin, U8 value, U8 value_hold, U8 *memory_map, U8 slot, U16
holdtime);
U8 Souliss_LowDigInHold(U8 pin, U8 value, U8 value_hold, U8 *memory_map, U8 slot, U16
holdtime);
void Souliss_ImportAnalog(U8* memory_map, U8 slot, float* analogvalue);
void Souliss_AnalogIn(U8 pin, U8 *memory_map, U8 slot, float scaling, float bias);
void Souliss_DigOut(U8 pin, U8 value, U8 *memory_map, U8 slot);
void Souliss_nDigOut(U8 pin, U8 value, U8 *memory_map, U8 slot);
void Souliss_LowDigOut(U8 pin, U8 value, U8 *memory_map, U8 slot);
```

```

void Souliss_nLowDigOut(U8 pin, U8 value, U8 *memory_map, U8 slot);
void Souliss_DigOutToggle(U8 pin, U8 value, U8 *memory_map, U8 slot);
void Souliss_DigOutLessThan(U8 pin, U8 value, U8 deadband, U8 *memory_map, U8 slot);
void Souliss_DigOutGreaterThan(U8 pin, U8 value, U8 deadband, U8 *memory_map, U8 slot);
void Souliss_LinkIO(U8 *memory_map, U8 input_slot, U8 output_slot, U8 *trigger);
void Souliss_LinkOI(U8 *memory_map, U8 input_slot, U8 output_slot);
void Souliss_ResetOutput(U8 *memory_map, U8 slot);
void Souliss_ResetInput(U8 *memory_map, U8 slot);
U8 Souliss_isTriggered(U8 *memory_map, U8 slot);

#if(HTTPSERVER && VNET_MEDIA1_ENABLE && (ETH_W5100 || ETH_W5200 || ETH_W5500))
#    include "interfaces/HTTP.h"
#elif(HTTPSERVER && VNET_MEDIA1_ENABLE && ETH_ENC28J60)
#    include "interfaces/HTTP_uIP.h"
#elif(ARDUINO_ETHLIB && VNET_MEDIA1_ENABLE && (ETH_W5100 || ETH_W5200 || ETH_W5500))
#    include "webhook/webhook.h"
#elif(XMLSERVER && VNET_MEDIA1_ENABLE && (ETH_W5100 || ETH_W5200 || ETH_W5500 || ETH_ENC28J60))
#    include "interfaces/XMLServer.h"
#elif(MODBUS)
#    include "interfaces/Modbus.h"
#endif

// Include IO definitions and drivers for supported hardware
#include "hardware/IOfdef.h"
#include "tools/IEEE754/float16.h"

// Some bytes in the EEPROM are reserved
#if(USEEEPROM)
#    include "tools/store/store.h"
#    include "tools/store/store.cpp"
#endif

#include "frame/MaCaco/MaCaco.cpp"
#include "frame/vNet/vNet.cpp"

#if(HTTPSERVER && VNET_MEDIA1_ENABLE && (ETH_W5100 || ETH_W5200 || ETH_W5500))
    #include "interfaces/HTTP.cpp"
#elif(HTTPSERVER && VNET_MEDIA1_ENABLE && ETH_ENC28J60)
    #include "interfaces/HTTP_uIP.cpp"
#elif((XMLSERVER == 1) && (VNET_MEDIA1_ENABLE && (ETH_W5100 || ETH_W5200 || ETH_W5500)))
    #    include "interfaces/XMLServer_HTTP.cpp"
#elif((XMLSERVER == 2) && (VNET_MEDIA1_ENABLE && (ETH_W5100 || ETH_W5200 || ETH_W5500)))
    #    include "interfaces/XMLServer_UDP.cpp"

```

```
#elif((XMLSERVER == 1) && (VNET_MEDIA1_ENABLE && ETH_ENC28J60))
#    include "interfaces/XMLServer_HTTP_uIP.cpp"
#elif(MODBUS)
#    include "interfaces/Modbus.cpp"
#endif

// Include IO definitions and drivers for supported hardware
#include "hardware/IOdef.cpp"

// Include methods for half-precision floating points
#include "tools/IEEE754/float16.c"

// Include Souliss code base and typicals
#include "base/Communication.cpp"
#include "base/LocalIO.cpp"
#include "base/NetworkSetup.cpp"
#include "base/T1n.cpp"
#include "base/T2n.cpp"
#include "base/T3n.cpp"
#include "base/T4n.cpp"
#include "base/T5n.cpp"

#include "tools/types.h"
#include "GetConfig.h"           // need : ethUsrCfg.h, vNetCfg.h, SoulissCfg.h,
MaCacoCfg.h

#include "base/PublishSubscribe.h"
#include "base/Sleeping.h"
#include "base/SpeakEasy.h"

#include "frame/MaCaco/MaCaco.h"
#include "frame/vNet/vNet.h"

// Include IO definitions and drivers for supported hardware
#include "hardware/IOdef.h"
#include "tools/IEEE754/float16.h"

#if (SOULISS_DEBUG)
    #define SOULISS_LOG Serial.print
#endif

#include "base/SpeakEasy.h"
#include "user/user_config.h"

#endif
```

**TYPICAL.H**

\*\*\*\*\*

Souliss  
Copyright (C) 2011 Veseo

This program 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 3 of the License, or (at your option) any later version.

This program 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 program. If not, see <<http://www.gnu.org/licenses/>>.

Originally developed by Dario Di Maio and Alessandro DelPex

```
*****  
/*!  
 \file  
 \ingroup  
 */  
#ifndef TYP_H  
#define TYP_H  
  
#include "tools/types.h"  
  
// Defines for Typicals  
#define Souliss_T1n 0x10 // Typicals  
Group 0x10  
#define Souliss_T2n 0x20 // Typicals  
Group 0x20  
#define Souliss_T3n 0x30 // Typicals  
Group 0x30  
#define Souliss_T4n 0x40 // Typicals  
Group 0x40  
#define Souliss_T5n 0x50 // Typicals  
Group 0x50  
  
#define Souliss_T11 0x11 // ON/OFF Digital  
Output with Timer Option  
#define Souliss_T12 0x12 // ON/OFF Digital  
Output with AUTO mode  
#define Souliss_T13 0x13 // Digital Input  
Value  
#define Souliss_T14 0x14 // Pulse Digital
```

**Output**

#define Souliss_T15	0x15	// RGB Light (IR)
#define Souliss_T16	0x16	// RGB Light
#define Souliss_T18	0x18	// ON/OFF Digital
Output with pulse output with Timer Option		
#define Souliss_T19	0x19	// LED Light
#define Souliss_T1A	0X1A	// Digital pass
through		
#define Souliss_T1B	0X1B	// Position
Constrained ON/OFF Digital Output		
#define Souliss_T21	0x21	// Motorized
devices with limit switches		
#define Souliss_T22	0x22	// Motorized
devices with limit switches and middle position		
#define Souliss_T31	0x31	// Temperature
control		
#define Souliss_T32	0x32	// Air Conditioner
#define Souliss_T41	0x41	// Anti-theft
integration (Main)		
#define Souliss_T42	0x42	// Anti-theft
integration (Peer)		
#define Souliss_T51	0x51	
#define Souliss_T52	0x52	
#define Souliss_T53	0x53	
#define Souliss_T54	0x54	
#define Souliss_TRL	0xFF	// Related
association, indentify a slot related to a previous one		

**// General defines for T1n**

#define Souliss_T1n_ToggleCmd	0x01	// Toggle Command
#define Souliss_T1n_OnCmd	0x02	// ON Command
#define Souliss_T1n_OffCmd	0x04	// OFF Command
#define Souliss_T1n_AutoCmd	0x08	// AUTO Mode Command
#define Souliss_T1n_BrightUp	0x10	// Increase Light
#define Souliss_T1n_BrightDown	0x20	// Decrease Light
#define Souliss_T1n_Flash	0x21	// Flash Light
#define Souliss_T1n_Set	0x22	// Set a state
#define Souliss_T1n_OnFeedback	0x23	// Report the actual state as ON
#define Souliss_T1n_OffFeedback	0x24	// Report the actual state as OFF
#define Souliss_T1n_Timed_StdVal	0x40	// Timed ON Standard Value
#define Souliss_T1n_Timed	0x30	// Timed ON
#define Souliss_T1n_PositionOnCmd	0x31	// Position constrained ON
Command		
#define Souliss_T1n_RstCmd	0x00	// Reset
#define Souliss_T1n_AutoState	0xF0	// AUTO Mode Feedback
#define Souliss_T1n_Coil	0x01	// Output Feedback ON
#define Souliss_T1n_OnCoil	0x01	// Output Feedback ON
#define Souliss_T1n_OffCoil	0x00	// Output Feedback OFF
#define Souliss_T1n_ResetCoil	0xA1	// Pulse Output Coil (Reset)

```

#define Souliss_T1n_PulseCoil          0xA5      // Pulse Output Coil (Set)
#define Souliss_T1n_TimedOnCoil        0xE1      // Output Feedback ON in Timed
Mode
#define Souliss_T1n_TimedOffCoil       0xE0      // Output Feedback OFF in Timed Mode
#define Souliss_T1n_AutoOnCoil         0xF1      // Output Feedback ON in AUTO
Mode
#define Souliss_T1n_AutoOffCoil        0xF0      // Output Feedback OFF in
AUTO Mode
#define Souliss_T1n_GoodNight          0xF1      // Output Feedback ON in
GoodNight Mode

#define Souliss_T1n_BrightValue        0x10

/*
// These values are not yet updated in SoulissApp, reverse to old ones
// thanks to Juan Pinto.

#define Souliss_T1n_RGBLamp_OnCmd     0x02      // ON Command
#define Souliss_T1n_RGBLamp_OffCmd    0x04      // OFF Command
#define Souliss_T1n_RGBLamp_RstCmd   0x00      // Reset Command
#define Souliss_T1n_RGBLamp_R         0x06      // Red Color
#define Souliss_T1n_RGBLamp_G         0x07      // Green Color
#define Souliss_T1n_RGBLamp_B         0x08      // Blue Color
#define Souliss_T1n_RGBLamp_W         0x09      // White Color
#define Souliss_T1n_RGBLamp_BrightUp 0x10
#define Souliss_T1n_RGBLamp_BrightDown 0x20
*/
#define Souliss_T1n_RGBLamp_OnCmd     0x01      // ON Command
#define Souliss_T1n_RGBLamp_OffCmd   0x09      // OFF Command
#define Souliss_T1n_RGBLamp_RstCmd   0x00      // Reset Command
#define Souliss_T1n_RGBLamp_R         0x02      // Red Color
#define Souliss_T1n_RGBLamp_G         0x03      // Green Color
#define Souliss_T1n_RGBLamp_B         0x04      // Blue Color
#define Souliss_T1n_RGBLamp_W         0x05      // White Color
#define Souliss_T1n_RGBLamp_BrightUp 0x06
#define Souliss_T1n_RGBLamp_BrightDown 0x07

#define Souliss_T1n_RGBLamp_Flash     0xA1
#define Souliss_T1n_RGBLamp_Strobe    0xA2
#define Souliss_T1n_RGBLamp_Fade      0xA3
#define Souliss_T1n_RGBLamp_Smooth    0xA4
#define Souliss_T1n_RGBLamp_R2         0xB1
#define Souliss_T1n_RGBLamp_R3         0xB2
#define Souliss_T1n_RGBLamp_R4         0xB3
#define Souliss_T1n_RGBLamp_R5         0xB4
#define Souliss_T1n_RGBLamp_G2         0xC1

```

```

#define Souliss_T1n_RGBLamp_G3          0xC2
#define Souliss_T1n_RGBLamp_G4          0xC3
#define Souliss_T1n_RGBLamp_G5          0xC4
#define Souliss_T1n_RGBLamp_B2          0xD1
#define Souliss_T1n_RGBLamp_B3          0xD2
#define Souliss_T1n_RGBLamp_B4          0xD3
#define Souliss_T1n_RGBLamp_B5          0xD4

// General defines for T2n
#define Souliss_T2n_CloseCmd_SW        0x01      // Close Command
#define Souliss_T2n_OpenCmd_SW         0x02      // Open Command
#define Souliss_T2n_CloseCmd           Souliss_T2n_CloseCmd_SW // Close
Command (legacy)
#define Souliss_T2n_OpenCmd            Souliss_T2n_OpenCmd_SW // Open Command (legacy)
#define Souliss_T2n_StopCmd           0x04      // Stop Command
#define Souliss_T2n_CloseCmd_Local    0x08      // Close Command (only from local pushbutton)
#define Souliss_T2n_OpenCmd_Local     0x10      // Open Command (only from local pushbutton)
#define Souliss_T2n_ToggleCmd         0x08      // Toggle Command
#define Souliss_T2n_RstCmd            0x00      // Reset Command
#define Souliss_T2n_Timer_Val         0xC0      // Timer set value
#define Souliss_T2n_Timer_Off         0xA0      // Timer expired value
#define Souliss_T2n_TimedStop_Val     0xC2      // Timed stop value
#define Souliss_T2n_TimedStop_Off     0xC0      // Timed stop exipred value
#define Souliss_T2n_LimSwitch_Close   0x08      // Close Feedback from Limit
Switch
#define Souliss_T2n_LimSwitch_Open     0x10      // Open Feedback from Limit
Switch
#define Souliss_T2n_NoLimSwitch       0x20      // No Limit Switch
#define Souliss_T2n_Coil_Close        0x01      // Closing
#define Souliss_T2n_Coil_Open         0x02      // Opening
#define Souliss_T2n_Coil_Stop         0x03      // Stopped
#define Souliss_T2n_Coil_Off          0x00      // Start state that will become
Souliss_T2n_Coil_Stop
#define Souliss_T2n_IsTemporaryStop   ((memory_map[MaCaco_AUXIN_s + slot] >
Souliss_T2n_TimedStop_Off) && (memory_map[MaCaco_AUXIN_s + slot] <=
Souliss_T2n_TimedStop_Val))

// General defines for T3n
#define Souliss_T3n_InSetPoint        0x01      // Increase Setpoint Command
#define Souliss_T3n_OutSetPoint        0x02      // Decrease Setpoint Command
#define Souliss_T3n_AsMeasuredMeasure 0x03      // Setpoint equal to actual
#define Souliss_T3n_Cooling           0x04      // Set cooling mode
#define Souliss_T3n_Heating            0x05      // Set heating mode
#define Souliss_T3n_FanOff             0x06      // Heating / Cooling Fan Off
#define Souliss_T3n_FanLow             0x07      // Heating / Cooling Fan

```

Low

#define Souliss_T3n_FanMed	0x08	// Heating / Cooling Fan
Medium		
#define Souliss_T3n_FanHigh	0x09	// Heating / Cooling Fan
High		
#define Souliss_T3n_FanAuto	0x0A	// Heating / Cooling Fan
Automatic		
#define Souliss_T3n_FanManual	0x0B	// Heating / Cooling Fan Manual
#define Souliss_T3n_DeadBand	0.01	// Percentage Deadband
#define Souliss_T3n_ThMed	0.10	// Threshold for medium speed
#define Souliss_T3n_ThHigh	0.15	// Threshold for high speed
#define Souliss_T3n_SetTemp	0x0C	// Set the setpoint
#define Souliss_T3n_ShutDown	0x0D	// Shut down heating and cooling

#define Souliss_T3n_RstCmd	0x0000
#define Souliss_T3n_HeatingOn	0x02 // Heating Active
#define Souliss_T3n_CoolingOn	0x04 // Cooling Active
#define Souliss_T3n_FanOn1	0x08 // Fan 1 Running
#define Souliss_T3n_FanOn2	0x10 // Fan 2 Running
#define Souliss_T3n_FanOn3	0x20 // Fan 3 Running
#define Souliss_T3n_FanAutoState	0x40 // Fan set in Automatic
#define Souliss_T3n_HeatingMode	0x80 // State set as heating
#define Souliss_T3n_CoolingMode	0x80 // State set as cooling

#define Souliss_T3n_AirCon_OnCmd	0xF0
#define Souliss_T3n_AirCon_OffCmd	0xFC
#define Souliss_T3n_AirCon_RstCmd	0x00
#define Souliss_T3n_AirCon_T16C	0x0F
#define Souliss_T3n_AirCon_T17C	0x07
#define Souliss_T3n_AirCon_T18C	0x0B
#define Souliss_T3n_AirCon_T19C	0x03
#define Souliss_T3n_AirCon_T20C	0x0D
#define Souliss_T3n_AirCon_T21C	0x05
#define Souliss_T3n_AirCon_T22C	0x09
#define Souliss_T3n_AirCon_T23C	0x01
#define Souliss_T3n_AirCon_T24C	0x0E
#define Souliss_T3n_AirCon_T25C	0x06
#define Souliss_T3n_AirCon_T26C	0x0A
#define Souliss_T3n_AirCon_T27C	0x02
#define Souliss_T3n_AirCon_T28C	0x0C
#define Souliss_T3n_AirCon_T29C	0x04
#define Souliss_T3n_AirCon_T30C	0x08
#define Souliss_T3n_AirCon_Normal	0x71
#define Souliss_T3n_AirCon_Eco	0x01
#define Souliss_T3n_AirCon_Turbo	0x11
#define Souliss_T3n_AirCon_Auto	0x0F
#define Souliss_T3n_AirCon_Dry	0x0B

```

#define Souliss_T3n_AirCon_Fan          0x03
#define Souliss_T3n_AirCon_Heat         0x0D
#define Souliss_T3n_AirCon_Cool        0x07
#define Souliss_T3n_AirCon_Fan_Auto    0x07
#define Souliss_T3n_AirCon_Fan_High   0x02
#define Souliss_T3n_AirCon_Fan_Med    0x06
#define Souliss_T3n_AirCon_Fan_Low    0x05
#define Souliss_T3n_AirCon_Opt1       0x2D
#define Souliss_T3n_AirCon_Opt2       0x77

// General defines for T4n
#define Souliss_T4n_Alarm             0x01      // Alarm Condition Detected
(Input)
#define Souliss_T4n_RstCmd            0x00
#define Souliss_T4n_ReArm             0x03      // Silence and Arm Command
#define Souliss_T4n_NotArmed          0x04      // Anti-theft not Armed
Command
#define Souliss_T4n_Armed             0x05      // Anti-theft Armed Command
#define Souliss_T4n_Antitheft         0x01      // Anti-theft Armed Feedback
#define Souliss_T4n_NoAntitheft       0x00      // Anti-theft not Armed Feedback
#define Souliss_T4n_InAlarm           0x03      // Anti-theft in Alarm

#define Souliss_RstCmd               0x00
#define Souliss_NOTTRIGGERED          0x00
#define Souliss_TRIGGERED              0x01

```

```

void Souliss_SetT11(U8 *memory_map, U8 slot);
U8 Souliss_Logic_T11(U8 *memory_map, U8 slot, U8 *trigger);
void Souliss_T11_Timer(U8 *memory_map, U8 input_slot);

```

```

void Souliss_SetT12(U8 *memory_map, U8 slot);
U8 Souliss_Logic_T12(U8 *memory_map, U8 slot, U8 *trigger);
void Souliss_T12_Timer(U8 *memory_map, U8 input_slot);

```

```

void Souliss_SetT13(U8 *memory_map, U8 slot);
U8 Souliss_Logic_T13(U8 *memory_map, U8 slot, U8 *trigger);

```

```

void Souliss_SetT14(U8 *memory_map, U8 slot);
U8 Souliss_Logic_T14(U8 *memory_map, U8 slot, U8 *trigger);

```

```

void Souliss_SetT15(U8 *memory_map, U8 slot);
U8 Souliss_Logic_T15(U8 *memory_map, U8 slot, U8 *trigger);

```

```

void Souliss_SetT16(U8 *memory_map, U8 slot);
U8 Souliss_Logic_T16(U8 *memory_map, U8 slot, U8 *trigger);
void Souliss_T16_Timer(U8 *memory_map, U8 input_slot);

```

```

void Souliss_SetT18(U8 *memory_map, U8 slot);

```

```

U8 Souliss_Logic_T18(U8 *memory_map, U8 slot, U8 *trigger);

void Souliss_SetT19(U8 *memory_map, U8 slot);
U8 Souliss_Logic_T19(U8 *memory_map, U8 slot, U8 *trigger);
void Souliss_T19_Timer(U8 *memory_map, U8 input_slot);

void Souliss_SetT1A(U8 *memory_map, U8 slot);
U8 Souliss_Logic_T1A(U8 *memory_map, U8 slot, U8 *trigger);

void Souliss_SetT21(U8 *memory_map, U8 slot);
U8 Souliss_Logic_T21(U8 *memory_map, U8 slot, U8 timeout);
void Souliss_T21_Timer(U8 *memory_map, U8 slot);

void Souliss_SetT22(U8 *memory_map, U8 slot);
U8 Souliss_Logic_T22(U8 *memory_map, U8 slot, U8 *trigger, U8 timeout);
void Souliss_T22_Timer(U8 *memory_map, U8 slot);

void Souliss_SetT31(U8 *memory_map, U8 slot);
U8 Souliss_Logic_T31(U8 *memory_map, U8 slot, U8 *trigger);

void Souliss_SetT32(U8 *memory_map, U8 slot);
U8 Souliss_Logic_T32(U8 *memory_map, U8 slot, U8 *trigger);

void Souliss_SetT41(U8 *memory_map, U8 slot);
U8 Souliss_Logic_T41(U8 *memory_map, U8 slot, U8 *trigger);
void Souliss_T41_Timer(U8 *memory_map, U8 slot);

void Souliss_SetT42(U8 *memory_map, U8 slot);
U8 Souliss_Logic_T42(U8 *memory_map, U8 slot, U8 *trigger, U16 main_antitheft_address);

void Souliss_SetT51(U8 *memory_map, U8 slot);
U8 Souliss_Logic_T51(U8 *memory_map, U8 slot, const float deadband, U8 *trigger);

void Souliss_SetT5n(U8 *memory_map, U8 slot, U8 typ);

#define Souliss_SetT52(memory_map, slot) Souliss_SetT5n(memory_map, slot, 0x52)
#define Souliss_SetT53(memory_map, slot) Souliss_SetT5n(memory_map, slot, 0x53)
#define Souliss_SetT54(memory_map, slot) Souliss_SetT5n(memory_map, slot, 0x54)
#define Souliss_SetT55(memory_map, slot) Souliss_SetT5n(memory_map, slot, 0x55)
#define Souliss_SetT56(memory_map, slot) Souliss_SetT5n(memory_map, slot, 0x56)
#define Souliss_SetT57(memory_map, slot) Souliss_SetT5n(memory_map, slot, 0x57)
#define Souliss_SetT58(memory_map, slot) Souliss_SetT5n(memory_map, slot, 0x58)
#define Souliss_SetT59(memory_map, slot) Souliss_SetT5n(memory_map, slot, 0x59)

#define Souliss_Logic_T52 Souliss_Logic_T51
#define Souliss_Logic_T53 Souliss_Logic_T51
#define Souliss_Logic_T54 Souliss_Logic_T51
#define Souliss_Logic_T55 Souliss_Logic_T51
#define Souliss_Logic_T56 Souliss_Logic_T51

```

```
#define Souliss_Logic_T57 Souliss_Logic_T51  
#define Souliss_Logic_T58 Souliss_Logic_T51  
#define Souliss_Logic_T59 Souliss_Logic_T51
```

```
#endif
```

## **GETCONFIG.H**

```
*****
```

Souliss  
Copyright (C) 2014 Veseo

This program 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 3 of the License, or  
(at your option) any later version.

This program 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 program. If not, see <<http://www.gnu.org/licenses/>>.

Originally developed by Dario Di Maio

```
*****  
/*!  
 \file  
 \ingroup  
 */  
*****  
#ifndef GENCONF_H  
#define GENCONF_H  
  
// Include configuration files  
#include "conf/frame/MaCacoCfg.h"  
#include "conf/uIP/uIPopt.h"  
#include "conf/chibi/chibiUsrCfg.h"  
#include "conf/nRF24/nRF24UsrCfg.h"  
#include "conf/RFM69/RFM69UsrCfg.h"  
#include "conf/usart/usartUsrCfg.h"  
  
#include "conf/eth/ethUsrCfg.h"  
#include "conf/hardware/hwBoards.h"  
#include "conf/frame/SoulissCfg.h"  
#include "conf/frame/vNetCfg.h"  
#include "user/user_config.h"  
  
// Define to zero if not used  
#ifndef VNET_MEDIA1_ENABLE  
#define VNET_MEDIA1_ENABLE 0  
#endif  
  
#ifndef VNET_MEDIA2_ENABLE
```

```
#define      VNET_MEDIA2_ENABLE 0
#endif

#ifndefVNET_MEDIA3_ENABLE
#define      VNET_MEDIA3_ENABLE 0
#endif

#ifndefVNET_MEDIA4_ENABLE
#define      VNET_MEDIA4_ENABLE 0
#endif

#ifndefVNET_MEDIA5_ENABLE
#define      VNET_MEDIA5_ENABLE 0
#endif

const U16 vnet_media_en[VNET_MEDIA_NUMBER] = {VNET_MEDIA1_ENABLE, // Media 1
                                              VNET_MEDIA2_ENABLE, // Media 2
                                              VNET_MEDIA3_ENABLE, // Media 3
                                              VNET_MEDIA4_ENABLE, // Media 4
                                              VNET_MEDIA5_ENABLE // Media 5
                                         };
#endif
```

**SPEAKEASY.H**


---

Souliss

Copyright (C) 2013 Veseo

This program 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 3 of the License, or (at your option) any later version.

This program 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 program. If not, see <<http://www.gnu.org/licenses/>>.

Originally developed by Dario Di Maio

---

```
/*
\file
\ingroup
*/
#ifndef SPEAKEASY_H
#define SPEAKEASY_H

#include "Typicals.h"

U8 memory_map[MaCaco_MEMMAP];           // define the shared memory map
U8 data_changed = 1;                    // flag

***** Scheduler *****
#define time_base_fast      10          // Time cycle in milliseconds
#define time_base_slow       10000       // Time cycle in milliseconds
#define num_phases           1000        // Number of phases

U16 phase_speedy=0, phase_fast=0, phase_slow=0;
unsigned long tmr_fast=0, tmr_slow=0;

#define EXECUTEFAST()  if(abs(millis()-tmr_fast) > time_base_fast)
#define UPDATEFAST()   tmr_fast = millis(); \
                           phase_fast = (phase_fast + 1) % num_phases

#define FAST_x10ms(n)     if (!(phase_fast % n))
#define FAST_10ms()        if (!(phase_fast % 1))
#define FAST_30ms()        if (!(phase_fast % 3))
```

```

#define FAST_50ms() if (!(phase_fast % 5))
#define FAST_70ms() if (!(phase_fast % 7))
#define FAST_90ms() if (!(phase_fast % 9))
#define FAST_110ms() if (!(phase_fast % 11))
#define FAST_510ms() if (!(phase_fast % 51))
#define FAST_710ms() if (!(phase_fast % 71))
#define FAST_910ms() if (!(phase_fast % 91))
#define FAST_1110ms() if (!(phase_fast % 111))
#define FAST_2110ms() if (!(phase_fast % 211))
#define FAST_7110ms() if (!(phase_fast % 711))
#define FAST_9110ms() if (!(phase_fast % 911))
#define FAST_11110ms() if (!(phase_fast % 1111))
#define FAST_21110ms() if (!(phase_fast % 2111))
#define FAST_71110ms() if (!(phase_fast % 7111))
#define FAST_91110ms() if (!(phase_fast % 9111))

#define EXECUTESLOW() else if(abs(millis()-tmr_slow) > time_base_slow)
#define UPDATESLOW() tmr_slow = millis(); \
                           phase_slow = (phase_slow + 1) % num_phases

#define SLOW_x10s(n) if (!(phase_slow % n))
#define SLOW_10s() if (!(phase_slow % 1))
#define SLOW_50s() if (!(phase_slow % 5))
#define SLOW_70s() if (!(phase_slow % 7))
#define SLOW_90s() if (!(phase_slow % 9))
#define SLOW_110s() if (!(phase_slow % 11))
#define SLOW_510s() if (!(phase_slow % 51))
#define SLOW_710s() if (!(phase_slow % 71))
#define SLOW_15m() if (!(phase_slow % 91))
#define SLOW_30m() if (!(phase_slow % 181))
#define SLOW_1h() if (!(phase_slow % 361))
#define SLOW_2h() if (!(phase_slow % 721))
#define SLOW_4h() if (!(phase_slow % 1441))
#define SLOW_halfday() if (!(phase_slow % 4321))
#define SLOW_1day() if (!(phase_slow % 8641))

#define EXECUTESPEEDY() else
#define UPDATESPEEDY() phase_speedy = (phase_speedy + 1) % num_phases
#define SPEEDY_x(n) if (!(phase_speedy % n))

/*****
***** Typicals *****/
#define Set_SimpleLight(slot) Souliss_SetT11(memory_map, slot)
#define Logic_SimpleLight(slot) Souliss_Logic_T11(memory_map, slot,
&data_changed)
#define Timer_SimpleLight(slot) Souliss_T11_Timer(memory_map, slot)

```

#define Set_T11(slot)	Souliss_SetT11(memory_map, slot)
#define Logic_T11(slot)	Souliss_Logic_T11(memory_map, slot,
&data_changed)	
#define Timer_T11(slot)	Souliss_T11_Timer(memory_map, slot)
#define Set_AutoLight(slot)	Souliss_SetT12(memory_map, slot)
#define Logic_AutoLight(slot)	Souliss_Logic_T12(memory_map, slot,
&data_changed)	
#define Timer_AutoLight(slot)	Souliss_T12_Timer(memory_map, slot)
#define Set_T12(slot)	Souliss_SetT12(memory_map, slot)
#define Logic_T12(slot)	Souliss_Logic_T12(memory_map, slot,
&data_changed)	
#define Timer_T12(slot)	Souliss_T12_Timer(memory_map, slot)
#define Set_DigitalInput(slot)	Souliss_SetT13(memory_map, slot)
#define Logic_DigitalInput(slot)	Souliss_Logic_T13(memory_map, slot,
&data_changed)	
#define Set_T13(slot)	Souliss_SetT13(memory_map, slot)
#define Logic_T13(slot)	Souliss_Logic_T13(memory_map, slot,
&data_changed)	
#define Set_PulseOutput(slot)	Souliss_SetT14(memory_map, slot)
#define Logic_PulseOutput(slot)	Souliss_Logic_T14(memory_map, slot,
&data_changed)	
#define Set_T14(slot)	Souliss_SetT14(memory_map, slot)
#define Logic_T14(slot)	Souliss_Logic_T14(memory_map, slot,
&data_changed)	
#define Set_IrDA_Lamp(slot)	Souliss_SetT15(memory_map, slot)
#define Logic_IrDA_Lamp(slot)	Souliss_Logic_T15(memory_map, slot,
&data_changed)	
#define Set_T15(slot)	Souliss_SetT15(memory_map, slot)
#define Logic_T15(slot)	Souliss_Logic_T15(memory_map, slot,
&data_changed)	
#define Set_LED_Strip(slot)	Souliss_SetT16(memory_map, slot)
#define Logic_LED_Strip(slot)	Souliss_Logic_T16(memory_map, slot,
&data_changed)	
#define Timer_LED_Strip(slot)	Souliss_T16_Timer(memory_map, slot)
#define Set_T16(slot)	Souliss_SetT16(memory_map, slot)
#define Logic_T16(slot)	Souliss_Logic_T16(memory_map, slot,
&data_changed)	
#define Timer_T16(slot)	Souliss_T16_Timer(memory_map, slot)
#define Set_DimmableLight(slot)	Souliss_SetT19(memory_map, slot)
#define Logic_DimmableLight(slot)	Souliss_Logic_T19(memory_map, slot, &data_changed)
#define Timer_DimmableLight(slot)	Souliss_T19_Timer(memory_map, slot)
#define Set_T19(slot)	Souliss_SetT19(memory_map, slot)
#define Logic_T19(slot)	Souliss_Logic_T19(memory_map, slot,

```

&data_changed)
#define Timer_T19(slot)           Souliss_T19_Timer(memory_map, slot)

#define Set_GarageDoor(slot)
#define Logic_GarageDoor(slot)
&data_changed)
#define Timer_GarageDoor(slot)     Souliss_SetT21(memory_map, slot)
#define Set_T21(slot)              Souliss_Logic_T21(memory_map, slot,
&data_changed)
#define Logic_T21(slot)            Souliss_T21_Timer(memory_map, slot)
#define Timer_T21(slot)             Souliss_SetT21(memory_map, slot,
&data_changed)
#define Set_Windows(slot)          Souliss_Logic_T21(memory_map, slot,
#define Logic_Windows(slot)        Souliss_T21_Timer(memory_map, slot)
&data_changed)
#define Timer_Windows(slot)        Souliss_SetT22(memory_map, slot)
#define Set_T22(slot)              Souliss_Logic_T22(memory_map, slot,
&data_changed)
#define Logic_T22(slot)            Souliss_T22_Timer(memory_map, slot)
#define Timer_T22(slot)             Souliss_SetT22(memory_map, slot)

#define Set_Thermostat(slot)       Souliss_SetT31(memory_map, slot)
#define Logic_Thermostat(slot)     Souliss_Logic_T31(memory_map, slot,
&data_changed)
#define Set_T31(slot)              Souliss_SetT31(memory_map, slot)
#define Logic_T31(slot)            Souliss_Logic_T31(memory_map, slot,
&data_changed)

#define Set_IrDA_Aircon(slot)      Souliss_SetT32(memory_map, slot)
#define Logic_IrDA_Aircon(slot)    Souliss_Logic_T32(memory_map, slot,
&data_changed)
#define Set_T32(slot)              Souliss_SetT32(memory_map, slot)
#define Logic_T32(slot)            Souliss_Logic_T32(memory_map, slot,
&data_changed)

#define Set_Antitheft_Main(slot)   Souliss_SetT41(memory_map, slot)
#define Logic_Antitheft_Main(slot) Souliss_Logic_T41(memory_map, slot,
&data_changed)
#define Timer_Antitheft_Main(slot) Souliss_T41_Timer(memory_map, slot)
#define Set_T41(slot)              Souliss_SetT41(memory_map, slot)
#define Logic_T41(slot)            Souliss_Logic_T41(memory_map, slot,
&data_changed)
#define Timer_T41(slot)            Souliss_T41_Timer(memory_map, slot)

#define Set_Antitheft_Peer(slot)   Souliss_SetT42(memory_map, slot)
#define Logic_Antitheft_Peer(slot, main_antitheft_address)
Souliss_Logic_T42(memory_map, slot, &data_changed, main_antitheft_address)
#define Set_T42(slot)

```

```

        Souliss_SetT42(memory_map, slot)
#define      Logic_T42(slot, main_antitheft_address)
        Souliss_Logic_T42(memory_map, slot, &data_changed, main_antitheft_address)

#define      Set_AnalogIn(slot)
#define      Read_AnalogIn(slot)
&data_changed)
#define      Set_T51(slot)
#define      Read_T51(slot)
0.05, &data_changed)

#define      Set_Temperature(slot)
#define      Logic_Temperature(slot)
&data_changed)
#define      Set_T52(slot)
#define      Logic_T52(slot)
0.05, &data_changed)

#define      Set_Humidity(slot)
#define      Logic_Humidity(slot)
&data_changed)
#define      Set_T53(slot)
#define      Logic_T53(slot)
0.05, &data_changed)

#define      Set_Light(slot)
#define      Logic_Light(slot)
&data_changed)
#define      Set_T54(slot)
#define      Logic_T54(slot)
0.05, &data_changed)

#define      Set_Voltage(slot)
#define      Logic_Voltage(slot)
&data_changed)
#define      Set_T55(slot)
#define      Logic_T55(slot)
0.05, &data_changed)

#define      Set_Current(slot)
#define      Logic_Current(slot)
&data_changed)
#define      Set_T56(slot)
#define      Logic_T56(slot)
0.05, &data_changed)

#define      Set_Power(slot)
#define      Logic_Power(slot)
&data_changed)

```

Souliss_SetT51(memory_map, slot)	
Souliss_Logic_T51(memory_map, slot, 0.05,	
Souliss_SetT51(memory_map, slot)	
Souliss_Logic_T51(memory_map, slot,	
Souliss_SetT52(memory_map, slot)	
Souliss_Logic_T52(memory_map, slot, 0.05,	
Souliss_SetT52(memory_map, slot)	
Souliss_Logic_T52(memory_map, slot,	
Souliss_SetT53(memory_map, slot)	
Souliss_Logic_T53(memory_map, slot, 0.05,	
Souliss_SetT53(memory_map, slot)	
Souliss_Logic_T53(memory_map, slot,	
Souliss_SetT54(memory_map, slot)	
Souliss_Logic_T54(memory_map, slot, 0.05,	
Souliss_SetT54(memory_map, slot)	
Souliss_Logic_T54(memory_map, slot,	
Souliss_SetT55(memory_map, slot)	
Souliss_Logic_T55(memory_map, slot, 0.05,	
Souliss_SetT55(memory_map, slot)	
Souliss_Logic_T55(memory_map, slot,	
Souliss_SetT56(memory_map, slot)	
Souliss_Logic_T56(memory_map, slot, 0.05,	
Souliss_SetT56(memory_map, slot)	
Souliss_Logic_T56(memory_map, slot,	
Souliss_SetT57(memory_map, slot)	
Souliss_Logic_T57(memory_map, slot, 0.05,	

```

#define      Set_T57(slot)           Souliss_SetT57(memory_map, slot)
#define      Logic_T57(slot)         Souliss_Logic_T57(memory_map, slot,
0.05, &data_changed)

#define      Set_Pressure(slot)      Souliss_SetT58(memory_map, slot)
#define      Logic_Pressure(slot)    Souliss_Logic_T58(memory_map, slot, 0.05,
&data_changed)
#define      Set_T58(slot)          Souliss_SetT58(memory_map, slot)
#define      Logic_T58(slot)        Souliss_Logic_T58(memory_map, slot,
0.05, &data_changed)

#define      Watchdog(chain_address, chain_slot, alarm_command)
Souliss_Watchdog(memory_map, chain_address, chain_slot, alarm_command)
/***********************/

***** Souliss *****/
#define myMap                      memory_map
#define myData                     &data_changed
#define myNode                    memory_map, &data_changed
#define ResetTrigger()            (data_changed=0)

#define SetAddress
Souliss_SetAddress
#define SetAddressingServer()
Souliss_SetAddressingServer(memory_map)
#define SetDynamicAddressing()
Souliss_SetDynamicAddressing()
#define SetIPAddress
Souliss_SetIPAddress
#define JoinNetwork()
Souliss_JoinNetwork()
#define JoinAndReset()
Souliss_JoinAndReset()
#define SetAsGateway(address)
Souliss_SetLocalAddress(memory_map, address)
#define SetAsPeerNode(address, index)
Souliss_SetRemoteAddress(memory_map, address, index)
#define SetAsBatteryNode(address, index)
Souliss_SetRemoteAddress(memory_map, address, index);      \
Souliss_BatteryChannels(memory_map, address)

#define HardcodedChannel
Souliss_HardcodedCommunicationChannel
#define GetTypicals()
Souliss_GetTypicals(memory_map)
#define CommunicationChannels()
Souliss_CommunicationChannels(memory_map)

```

```

#define ProcessCommunication()
    Souliss_CommunicationData(memory_map, &data_changed)
#define DigIn(pin,value,slot)
    Souliss_DigIn(pin, value, memory_map, slot)
#define LowDigIn(pin,value,slot)
    Souliss_LowDigIn(pin, value, memory_map, slot)
#define DigIn2State(pin,value_state_on,value_state_off,slot)      Souliss_DigIn2State(pin,
value_state_on, value_state_off, memory_map, slot)
#define DigInHold(pin, value_state1,value_state2,slot)
    Souliss_DigInHold(pin, value_state1, value_state2, memory_map, slot)
#define LowDigInHold(pin, value_state1,value_state2,slot)
    Souliss_LowDigInHold(pin, value_state1, value_state2, memory_map, slot)
#define DigOut(pin,value,slot)
    Souliss_DigOut(pin, value, memory_map, slot)
#define nDigOut(pin,value,slot)
    Souliss_nDigOut(pin, value, memory_map, slot)
#define LowDigOut(pin,value,slot)
    Souliss_LowDigOut(pin, value, memory_map, slot)
#define nLowDigOut(pin,value,slot)
    Souliss_nLowDigOut(pin, value, memory_map, slot)
#define DigOutToggle(pin,value,slot)
    Souliss_DigOutToggle(pin, value, memory_map, slot)
#define ImportAnalog(slot,analogvalue)
    Souliss_ImportAnalog(memory_map, slot, analogvalue)
#define AnalogIn(pin, slot, conv, bias)
    Souliss_AnalogIn(pin, memory_map, slot, conv, bias)
#define isTriggered(slot)
    Souliss_isTriggered(memory_map, slot)

#define RemoteInput
    Souliss_RemoteInput
#define RemoteInputs
    Souliss_RemoteInputs
#define RoutingTable
    vNet_SetRoutingTable
#define Init_XMLServer()
    XMLSERVERInit(memory_map)
#define Run_XMLServer()
    XMLSERVERInterface(memory_map)
#define Init_Modbus()
    ModbusInit(memory_map)
#define Run_Modbus()
    Modbus(memory_map)
#define Init_HTTPServer()
    {}
#define Run_HTTPServer()
    HTTPServer(memory_map)

```



```

        }

#define      GetAddress()
    Souliss_DynamicAddressing_FirstBoot (memory_map);
    \
n=0;n<VNET_MEDIA_NUMBER;n++) {
    \
        for(uint8_t
            \
            \
while(Souliss_DynamicAddressing (memory_map, __TIME__, 9)) {
    \
        for(U16 i=0; i<1000; i++) {
            \
                if(Souliss_CommunicationData(memory_map, &data_changed)) \
                    \
                        break;
                \
delay(10);      }
            \
        }
    \
while(!MaCaco_IsSubscribed()) {
    \
        Souliss_JoinAndReset();
    \
        \
for(U16 i=0; i<1000; i++) {
    \
        if(Souliss_CommunicationData(memory_map, &data_changed)) \
            \
                break;
        \
delay(10);      }
    \
}}}

```

```
#define WaitSubscription()
{      \

```

```
        while(!MaCaco_IsSubscribed())

```

```

ProcessCommunication();           \
delay(100);}

#define      aMinuteToSleep()          for(uint8_t s=0; s<255; s+
+)   {   \
ProcessCommunication();           \
delay(200);}

/****************************************/
/****************************************/
/*!
 Macros
*/
/****************************************/
#define AUX          MaCaco_AUXIN_s
#define IN           MaCaco_IN_s
#define TYP          MaCaco_TYP_s
#define OUT          MaCaco_OUT_s
#define mAuxiliary(slot) memory_map[AUX+slot]
#define mInput(slot)  memory_map[IN+slot]
#define mTypical(slot) memory_map[TYP+slot]
#define mOutput(slot) memory_map[OUT+slot]
#define mOutputAsFloat(node,slot) returnfloat32(&(memory_map[OUT+slot]))

#define pTYP          MaCaco_P_TYP_s
#define pOUT          MaCaco_P_OUT_s
#define pTypical(node,slot) memory_map[pTYP+slot+
(node*MaCaco_SLOT)]
#define pOutput(node,slot) memory_map[pOUT+slot+
(node*MaCaco_SLOT)]
#define pOutputAsFloat(node,slot) returnfloat32(&(memory_map[pOUT+slot+
(node*MaCaco_SLOT)]))

#define SetTrigger()    data_changed=1;
#define ResetTrigger() data_changed=0;
#define Initialize()   MaCaco_init(memory_map)
#define isTrigger()     data_changed

#endif

```

## T1N.CPP

```
*****
```

Souliss  
Copyright (C) 2011 Veseo

This program 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 3 of the License, or (at your option) any later version.

This program 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 program. If not, see <<http://www.gnu.org/licenses/>>.

Originally developed by Dario Di Maio and Alessandro DelPex

```
*****
```

```
/*!
```

```
\file  
\ingroup
```

```
*/
```

```
*****
```

```
/*!
```

Define the use of Typical 11 : ON/OFF Digital Output with Timer Option

```
*/
```

```
*****
```

```
void Souliss_SetT11(U8 *memory_map, U8 slot)
```

```
{
```

```
    memory_map[MaCaco_TYP_s + slot] = Souliss_T11;
```

```
}
```

```
*****
```

```
/*!
```

Typical 11 : ON/OFF Digital Output with Timer Option

Handle one digital output based on hardware and software commands, output can be timed out.

This logic can be used for lights, wall socket and all the devices that has an ON/OFF behaviour.

Hardware Command:

If using a monostable wall switch (press and spring return), each press will toggle the output status.

```
#define Souliss_T1n_ToggleCmd           0x01
```

If using a bistable wall switch (press without return), the two switch position can be associated with the ON and OFF commands

```
#define Souliss_T1n_OnCmd              0x02
#define Souliss_T1n_OffCmd              0x04
```

#### Software Commands:

From any available software interface, these commands will turn the light ON and OFF.

```
#define Souliss_T1n_OnCmd              0x02
#define Souliss_T1n_OffCmd              0x04
```

#### Timed Output Command:

This logic can be used with a timer to active the output state for a predefined amount of time, timer is activated if the input command is greater than

```
#define Souliss_T1n_Timed             0x30
```

A timed out command can be sent via software or hardware commands, in case of hardware commands is usually associated to a long press of a monostable wall switch.

If INPUTVAL is the input value the output will be timed for nCYCLES of the associated timer.

```
nCYCLES = INPUTVAL - Souliss_T1n_Timed
```

#### Command recap, using:

- 1(hex) as command, toggle the output
- 2(hex) as command, the output move to ON
- 4(hex) as command, the output move to OFF
- >30(hex) as command, time the output to ON
- 0(hex) as command, no action

#### Output status,

- 0(hex) for output OFF,
- 1(hex) for output ON.

```
*/
*****
U8 Souliss_Logic_T11(U8 *memory_map, U8 slot, U8 *trigger)
{
    U8 i_trigger=0;
        // Internal trigger
```

```

// Look for input value, update output. If the output is not set, trig a data
// change, otherwise just reset the input

if(memory_map[MaCaco_IN_s + slot] > Souliss_T1n_Timed) // Memory value is used as
timer
{
    if(memory_map[MaCaco_OUT_s + slot] != Souliss_T1n_OnCoil)
        i_trigger = Souliss_TRIGGERED;
    // Trig change

        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OnCoil; // Switch on the output
    memory_map[MaCaco_AUXIN_s + slot] = memory_map[MaCaco_IN_s + slot];
//Set the timer value
    memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd; // Reset
}

else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_OffCmd ||
         memory_map[MaCaco_AUXIN_s + slot] ==
Souliss_T1n_Timed) // Off Command
{
    if(memory_map[MaCaco_OUT_s + slot] != Souliss_T1n_OffCoil)
        i_trigger = Souliss_TRIGGERED;

        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OffCoil; // Switch off the output
    memory_map[MaCaco_AUXIN_s + slot] = 0; //Reset the timer
    memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd; // Reset
}

else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_OnCmd)
{
    if(memory_map[MaCaco_OUT_s + slot] != Souliss_T1n_OnCoil)
        i_trigger = Souliss_TRIGGERED;

        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OnCoil; // Switch on the output
    memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd; // Reset
}

else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_ToggleCmd)
{
    if(memory_map[MaCaco_OUT_s + slot] == Souliss_T1n_OnCoil)
        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OffCoil; // Switch OFF the output
    else if(memory_map[MaCaco_OUT_s + slot] == Souliss_T1n_OffCoil)
        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OnCoil; // Switch ON the output
}

```

```

        i_trigger = Souliss_TRIGGERED;
        memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;           // 
Reset
    }

    // Update the trigger
    if(i_trigger)
        *trigger = i_trigger;

    return i_trigger;
}

//****************************************************************************
/*!
    Timer associated to T11
*/
//****************************************************************************
void Souliss_T11_Timer(U8 *memory_map, U8 input_slot)
{
    if(memory_map[MaCaco_AUXIN_s + input_slot] > Souliss_T1n_Timed)
        // Memory value is used as timer
        memory_map[MaCaco_AUXIN_s + input_slot]--;
        // Decrease timer
}

//****************************************************************************
/*!
    Define the use of Typical 12 : ON/OFF Digital Output with AUTO mode
*/
//****************************************************************************
void Souliss_SetT12(U8 *memory_map, U8 slot)
{
    memory_map[MaCaco_TYP_s + slot] = Souliss_T12;
}

//****************************************************************************
/*!
    Typical 12 : ON/OFF Digital Output with AUTO mode

```

Handle one digital output based on hardware and software commands,  
output can be set in AUTO mode.

This logic can be used for lights, wall socket and all the devices  
that has an ON/OFF behaviour, output can be driven by external event  
like a PIR sensor.

Hardware Command:

If using a monostable wall switch (press and spring return),

each press will toggle the output status.

#define Souliss_T1n_ToggleCmd	0x01
-------------------------------	------

If using a bistable wall switch (press without return), the two switch position can be associated with the ON and OFF commands

#define Souliss_T1n_OnCmd	0x02
#define Souliss_T1n_OffCmd	0x04

Every time that an ON, OFF or TOOGLE command is received the AUTO mode is automatically removed.

#### Software Commands:

From any available software interface, these commands will turn the light ON and OFF.

#define Souliss_T1n_OnCmd	0x02
#define Souliss_T1n_OffCmd	0x04

Every time that an ON, OFF or TOOGLE command is received the AUTO mode is automatically removed.

#### AUTO Mode:

If in AUTO the output is driven by external event, such events has no effect if the AUTO mode is not active.

This logic MUST be used with a timer to active the output state for a predefined amount of time, once the external event is recognized. Once the external event occur, the input of the logic shall be greater than

#define Souliss_T1n_AutoCmd	0x08
-----------------------------	------

If INPUTVAL is the input value associated to the external event the output will be timed for nCYCLES of the associated timer.

nCYCLES = INPUTVAL - Souliss\_T1n\_Timed

#### Command recap, using:

- 1(hex) as command, toggle the output
- 2(hex) as command, the output move to ON the mode is reset
- 4(hex) as command, the output move to OFF the mode is reset
- 8(hex) as command, the mode is set to AUTO
- >8(hex) as command, time the output to ON if in AUTO
- 0(hex) as command, no action

#### Output status,

- 0(hex) for output OFF and mode not in AUTO,
- 1(hex) for output ON and mode not in AUTO,
- F0(hex) for output OFF and mode in AUTO,
- F1(hex) for output ON and mode in AUTO.

```
/*
*****
U8 Souliss_Logic_T12(U8 *memory_map, U8 slot, U8 *trigger)
{
    U8 i_trigger=0;
        // Internal trigger

    // Look for input value, update output. If the output is not set, trig a data
    // change, otherwise just reset the input

    if((memory_map[MaCaco_IN_s + slot] > Souliss_T1n_AutoCmd) &&
(memory_map[MaCaco_OUT_s + slot] & Souliss_T1n_AutoState)) // Memory value is used as
timer
    {
        if((memory_map[MaCaco_OUT_s + slot] & ~Souliss_T1n_AutoState) !=

Souliss_T1n_OnCoil)
            i_trigger = Souliss_TRIGGERED;
        // Trig change

        memory_map[MaCaco_OUT_s + slot] = (Souliss_T1n_AutoState |
Souliss_T1n_OnCoil); // Switch on the output
    }
    else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_OffCmd) // Off
Command
    {
        if((memory_map[MaCaco_OUT_s + slot] & ~Souliss_T1n_AutoState) !=

Souliss_T1n_OffCoil)
            i_trigger = Souliss_TRIGGERED;

        // Switch OFF and reset the AUTO mode
        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OffCoil; // Switch off the output and reset the AUTO mode
        memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd; // Reset
    }
    else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_OnCmd)
    {
        if((memory_map[MaCaco_OUT_s + slot] & ~Souliss_T1n_AutoState) !=

Souliss_T1n_OnCoil)
            i_trigger = Souliss_TRIGGERED;

        // Switch ON and reset the AUTO mode
        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OnCoil; // Switch on the output
        memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd; // Reset
    }
    else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_ToggleCmd)
```

```

{
    if((memory_map[MaCaco_OUT_s + slot] & ~Souliss_T1n_AutoState) ==
Souliss_T1n_OnCoil)
        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OffCoil;           //
Switch OFF the output and reset the AUTO mode
    else if((memory_map[MaCaco_OUT_s + slot] & ~Souliss_T1n_AutoState) ==
Souliss_T1n_OffCoil)
        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OnCoil;           //
Switch ON the output and reset the AUTO mode

    i_trigger = Souliss_TRIGGERED;
    // Trig the state change
    memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;               //
Reset
}
else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_AutoCmd)          // Set
AUTO state
{
    // If the logic is not in AUTO, active AUTO mode
    if((memory_map[MaCaco_OUT_s + slot] == (Souliss_T1n_AutoState |
Souliss_T1n_Coil)) || ((memory_map[MaCaco_OUT_s + slot] & Souliss_T1n_AutoState) !=
Souliss_T1n_AutoState))
        memory_map[MaCaco_OUT_s + slot] = (Souliss_T1n_AutoState |           //
Souliss_T1n_OffCoil);                                              // Switch OFF and set the AUTO mode
    else
        memory_map[MaCaco_OUT_s + slot] = (~Souliss_T1n_AutoState &           //
Souliss_T1n_OffCoil);                                              // Switch OFF and reset the AUTO mode

    i_trigger = Souliss_TRIGGERED;
    memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;               //
Reset
}
else
    memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;               //
No recognized command, reset

    // Update the trigger
    if(i_trigger)
        *trigger = i_trigger;

    return i_trigger;
}

*****
*/
Timer associated to T12
*/
*****
void Souliss_T12_Timer(U8 *memory_map, U8 input_slot)

```

```
{
    if(memory_map[MaCaco_IN_s + input_slot] > Souliss_T1n_AutoCmd)           // Memory
value is used as timer
        memory_map[MaCaco_IN_s + input_slot]--;
        // Decrease timer
}

/*****************************************/
/*!
 Define the use of Typical 13 : Digital Input Value
*/
/*****************************************/
void Souliss_SetT13(U8 *memory_map, U8 slot)
{
    memory_map[MaCaco_TYP_s + slot] = Souliss_T13;
}

/*****************************************/
/*!
 Typical 13 : Digital Input Value

```

It read data from input are of the memory map and place the values in the output, this logic shall be used to monitor a digital value into an user interface.

Hardware Command:

Using a bistable input switch (press without return) the two switch position can be associated with the ON and OFF commands

#define Souliss_T1n_OnCmd	0x02
#define Souliss_T1n_OffCmd	0x04

Command recap, using:

- 2(hex) as command, the output move to ON
- 4(hex) as command, the output move to OFF
- 0(hex) as command, no action

Output status,

- 0(hex) for output OFF,
- 1(hex) for output ON.

```
/*
*****************************************/
U8 Souliss_Logic_T13(U8 *memory_map, U8 slot, U8 *trigger)
{
    U8 i_trigger=0;
        // Internal trigger

        // Look for input value, update output. If the output is not set, trig a data

```

```

// change, otherwise just reset the input

    if(memory_map[MaCaco_IN_s + slot] == Souliss_T1n_OnCmd) // Memory value is used as timer
    {
        if(memory_map[MaCaco_OUT_s + slot] != Souliss_T1n_OnCoil)
            i_trigger = Souliss_TRIGGERED;
        // Trig change

        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OnCoil;
        memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd; // Reset
    }

    else if(memory_map[MaCaco_IN_s + slot] == Souliss_T1n_OffCmd) // Memory value is used as timer
    {
        if(memory_map[MaCaco_OUT_s + slot] != Souliss_T1n_OffCoil)
            i_trigger = Souliss_TRIGGERED;
        // Trig change

        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OffCoil;
        memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd; // Reset
    }

    // Update the trigger
    if(i_trigger)
        *trigger = i_trigger;

    return i_trigger;
}

//****************************************************************************
/*!
 Define the use of Typical 14 : Pulse Digital Output
*/
//****************************************************************************
void Souliss_SetT14(U8 *memory_map, U8 slot)
{
    memory_map[MaCaco_TYP_s + slot] = Souliss_T14;
}

//****************************************************************************
/*!
 Typical 14 : Pulse Digital Output

```

It control a digital output with a pulse of the duration of one execution cycle.

Hardware and/or Software Command:

Using a monostable input switch (press with return) or a software command from the user interface to have a pulse on the output  
 #define Souliss\_T1n\_OnCmd 0x02

Command recap, using:

- 2(hex) as command, the output move to ON for a cycle
- 0(hex) as command, no action

Output status,

- 0(hex) for output OFF,
- 1(hex) for output ON pulse.

```
/*
 ****
U8 Souliss_Logic_T14(U8 *memory_map, U8 slot, U8 *trigger)
{
    U8 i_trigger=0;
                    // Internal trigger

    // If the ON command is received
    if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_OnCmd)
    {
        // Set the output ON
        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OnCoil;

        // Reset the command
        memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;
        i_trigger = Souliss_TRIGGERED;
    }
    else // Set the output OFF
        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OffCoil;

    // Update the trigger
    if(i_trigger)
        *trigger = i_trigger;

    return i_trigger;
}

/*
*/
/*!
 Define the use of Typical 15 : RGB Light
*/
****

void Souliss_SetT15(U8 *memory_map, U8 slot)
{
```

```

memory_map[MaCaco_TYP_s + slot] = Souliss_T15;
memory_map[MaCaco_TYP_s + slot + 1] = Souliss_TRL;
}

//****************************************************************************
/*!
```

### Typical 15 : RGB Light

It map output over inputs matching the user interface commands, act as a remote control for the conditioner and require a mapping of the typical commands and the expected commands on the conditioner.

#### Software Commands:

A command is long one byte and indicate one operating mode of the RGB light. In the definitions are available the commands that shall be mapped versus the RGB light command map.

It use two memory slots

```

*/
//****************************************************************************
U8 Souliss_Logic_T15(U8 *memory_map, U8 slot, U8 *trigger)
{
    U8 i_trigger=0;
                // Internal trigger

    // Before power OFF store the last state
    if(memory_map[MaCaco_IN_s + slot] == Souliss_T1n_RGBLamp_OffCmd )
        memory_map[MaCaco_AUXIN_s + slot + 1] = memory_map[MaCaco_OUT_s +
slot];

    // If previosly there was an ON command, set the last state
    if((memory_map[MaCaco_OUT_s + slot] == Souliss_T1n_RGBLamp_OnCmd) &&
(memory_map[MaCaco_AUXIN_s + slot + 1]))
    {
        memory_map[MaCaco_OUT_s + slot] = memory_map[MaCaco_AUXIN_s + slot +
1];

        // Flag the change in the state
        memory_map[MaCaco_AUXIN_s + slot] = Souliss_TRIGGERED;
        i_trigger = Souliss_TRIGGERED;
    }
    else if(memory_map[MaCaco_IN_s + slot] != Souliss_T1n_RGBLamp_RstCmd)
    {
        // Use the incoming command
        memory_map[MaCaco_OUT_s + slot] = memory_map[MaCaco_IN_s + slot];

        // Flag the change in the state
    }
}
```

```

        memory_map[MaCaco_AUXIN_s + slot] = Souliss_TRIGGERED;
        i_trigger = Souliss_TRIGGERED;

        memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RGBLamp_RstCmd;
        // Reset
    }

/*
At this stage the requested command is available in the outputs, a mapping between
these command and what is expected by the air conditioner device shall be used just
after this method.
*/
// Update the trigger
if(i_trigger)
    *trigger = i_trigger;

return i_trigger;
}

*****
/*!
Define the use of Typical 16 : RGB LED Strip
*/
*****
void Souliss_SetT16(U8 *memory_map, U8 slot)
{
    memory_map[MaCaco_TYP_s + slot] = Souliss_T16;
    memory_map[MaCaco_TYP_s + slot + 1] = Souliss_TRL;
    memory_map[MaCaco_TYP_s + slot + 2] = Souliss_TRL;
    memory_map[MaCaco_TYP_s + slot + 3] = Souliss_TRL;
}

*****
/*!
Typical 16 : RGB LED Strip

```

Handle one RGB LED strip based on software commands, with automatic fade effects and "Good Night" feature. It use four (4) SLOT.

#### Hardware Command:

If using a monostable wall switch (press and spring return), each press will toggle the output status.

#define Souliss_T1n_ToggleCmd	0x01
#define Souliss_T1n_BrightUp	0x10
#define Souliss_T1n_BrightDown	0x20

If using a bistable wall switch (press without return), the two switch position can be associated with the ON and OFF commands

#define Souliss_T1n_OnCmd	0x02
#define Souliss_T1n_OffCmd	0x03

#### Software Commands:

From any available software interface, these commands will turn the light ON and OFF.

#define Souliss_T1n_OnCmd	0x02
#define Souliss_T1n_OffCmd	0x03
#define Souliss_T1n_BrightUp	0x10
#define Souliss_T1n_BrightDown	0x20
#define Souliss_T1n_Flash	0x21

Color can be set using the following command in the first slot, and giving the R, B and G values into the next three slots

#define Souliss_T1n_Set	0x22
-------------------------	------

#### Good Night:

The light goes off with a long time fade effect, this feature needs the external timer. Good Night mode is activated using if the input command is greater than

#define Souliss_T1n_Timed	0x30
---------------------------	------

A Good Night command can be sent via software or hardware commands, in case of hardware commands is usually associated to a long press of a monostable wall switch.

If INPUTVAL is the input value the output will be timed for nCYCLES of the associated timer.

nCYCLES = INPUTVAL - Souliss_T1n_Timed
--

#### Command recap, using:

- 1(hex) as command, toggle the output
- 2(hex) as command, the output move to ON
- 4(hex) as command, the output move to OFF
- 10(hex) as command, the light bright is increased
- 20(hex) as command, the light bright is decreased
- 21(hex) as command, the light flash on and off at each cycle
- 22(hex) as command, set the color (needs R, G and B values)
- >30(hex) as command, time the output to ON
- 0(hex) as command, no action

#### Output status,

- 0(hex) for output OFF,
- 1(hex) for output ON.

The next three slots contains the R, G and B values applied to the light.

```
/*
*****
U8 Souliss_Logic_T16(U8 *memory_map, U8 slot, U8 *trigger)
{
    U8 i_trigger=0;
        // Internal trigger

    // Look for input value, update output. If the output is not set, trig a data
    // change, otherwise just reset the input

    if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_ToggleCmd)           // Toggle
Command
    {
        // Toggle the actual status of the light
        if(memory_map[MaCaco_OUT_s + slot] == Souliss_T1n_OffCoil)
            memory_map[MaCaco_IN_s + slot] = Souliss_T1n_OnCmd;

        else if(memory_map[MaCaco_OUT_s + slot] == Souliss_T1n_OnCoil ||
memory_map[MaCaco_OUT_s + slot] == Souliss_T1n_GoodNight )
            memory_map[MaCaco_IN_s + slot] = Souliss_T1n_OffCmd;
        else
            memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;

        // Trig the change
        i_trigger = Souliss_TRIGGERED;
    }
    else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_OffCmd)           // Off
Command
    {
        // Trigger the change and save the actual color
        if(memory_map[MaCaco_OUT_s + slot] != Souliss_T1n_OffCoil)
        {
            memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OffCoil;          //
Switch off the light state
            i_trigger = Souliss_TRIGGERED;
            // Trig the change
        }

        // Fade out and turn off the light step wise
        for(U8 i=1;i<4;i++)
            if(memory_map[MaCaco_OUT_s + slot + i])
                memory_map[MaCaco_OUT_s + slot + i]--;
    }

    // Once is off, reset
    if((memory_map[MaCaco_OUT_s + slot + 1] == 0) &&

```

```

(memory_map[MaCaco_OUT_s + slot + 2] == 0)
    && (memory_map[MaCaco_OUT_s + slot + 3] == 0))
        memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;           //
Reset

    }

else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_OnCmd)
{
    if(memory_map[MaCaco_OUT_s + slot] != Souliss_T1n_OnCoil)
        i_trigger = Souliss_TRIGGERED;

    memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OnCoil;           //

Switch on the output

    // If there were no color set, use a light white
    if((memory_map[MaCaco_AUXIN_s + slot + 1] == 0) &&
(memory_map[MaCaco_AUXIN_s + slot + 2] == 0)
    && (memory_map[MaCaco_AUXIN_s + slot + 3] == 0))
    {
        for(U8 i=1;i<4;i++)
        {
            memory_map[MaCaco_OUT_s + slot + i] = 0xAA;
        // Set a light white
            memory_map[MaCaco_AUXIN_s + slot + i] =
memory_map[MaCaco_OUT_s + slot + i];
        }

        memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;
    // Reset
    }
    else
    {
        // Fade in to the last color
        for(U8 i=1;i<4;i++)
            if(memory_map[MaCaco_OUT_s + slot + i] <
(memory_map[MaCaco_AUXIN_s + slot + i]))
                memory_map[MaCaco_OUT_s + slot + i]++;
    }

    // Once is on, reset
    if((memory_map[MaCaco_OUT_s + slot + 1] == memory_map[MaCaco_AUXIN_s
+ slot + 1])
    && (memory_map[MaCaco_OUT_s + slot + 2] ==
memory_map[MaCaco_AUXIN_s + slot + 2])
    && (memory_map[MaCaco_OUT_s + slot + 3] ==
memory_map[MaCaco_AUXIN_s + slot + 3]))
    {
        memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;
    }
}

```

```

// Reset
    }
}
else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_BrightUp)      // Increase
the light value
{
    // Increase the light value
    if(memory_map[MaCaco_OUT_s + slot + 1] < 255 - Souliss_T1n_BrightValue)
        memory_map[MaCaco_OUT_s + slot + 1] += Souliss_T1n_BrightValue;

    if(memory_map[MaCaco_OUT_s + slot + 2] < 255 - Souliss_T1n_BrightValue)
        memory_map[MaCaco_OUT_s + slot + 2] += Souliss_T1n_BrightValue;

    if(memory_map[MaCaco_OUT_s + slot + 3] < 255 - Souliss_T1n_BrightValue)
        memory_map[MaCaco_OUT_s + slot + 3] += Souliss_T1n_BrightValue;

    // Save the new output value
    for(U8 i=1;i<4;i++)
        memory_map[MaCaco_AUXIN_s + slot + i] =
memory_map[MaCaco_AUXIN_s + slot + i];

    memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;                      //

Reset
}
else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_BrightDown)
    // Decrease the light value
{
    // Decrease the light value
    if(memory_map[MaCaco_OUT_s + slot + 1] > Souliss_T1n_BrightValue)
        memory_map[MaCaco_OUT_s + slot + 1] -= Souliss_T1n_BrightValue;

    if(memory_map[MaCaco_OUT_s + slot + 2] > Souliss_T1n_BrightValue)
        memory_map[MaCaco_OUT_s + slot + 2] -= Souliss_T1n_BrightValue;

    if(memory_map[MaCaco_OUT_s + slot + 3] > Souliss_T1n_BrightValue)
        memory_map[MaCaco_OUT_s + slot + 3] -= Souliss_T1n_BrightValue;

    // Save the new output value
    for(U8 i=1;i<4;i++)
        memory_map[MaCaco_AUXIN_s + slot + i] =
memory_map[MaCaco_AUXIN_s + slot + i];

    memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;                      //

Reset
}
else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_Flash)
    // Turn ON and OFF at each cycle
{
    // If the light was on

```

```

if(memory_map[MaCaco_OUT_s + slot] == Souliss_T1n_OnCoil)
{
    // Switch ON and OFF until a new command is received
    if((memory_map[MaCaco_OUT_s + slot + 1] ==
memory_map[MaCaco_AUXIN_s + slot + 1])
        && (memory_map[MaCaco_OUT_s + slot + 2] ==
memory_map[MaCaco_AUXIN_s + slot + 2])
        && (memory_map[MaCaco_OUT_s + slot + 3] ==
memory_map[MaCaco_AUXIN_s + slot + 3]))
    {
        // Set output to zero
        memory_map[MaCaco_OUT_s + slot + 1] = 0;
        memory_map[MaCaco_OUT_s + slot + 2] = 0;
        memory_map[MaCaco_OUT_s + slot + 3] = 0;
    }
    else if((memory_map[MaCaco_OUT_s + slot + 1] == 0)
        && (memory_map[MaCaco_OUT_s + slot + 2] == 0)
        && (memory_map[MaCaco_OUT_s + slot + 3] == 0))
    {
        // Set output to previous value
        memory_map[MaCaco_OUT_s + slot + 1] =
memory_map[MaCaco_AUXIN_s + slot + 1];
        memory_map[MaCaco_OUT_s + slot + 2] =
memory_map[MaCaco_AUXIN_s + slot + 2];
        memory_map[MaCaco_OUT_s + slot + 3] =
memory_map[MaCaco_AUXIN_s + slot + 3];
    }
}
else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_Set)
{
    // Set the new color
    for(U8 i=1;i<4;i++)
    {
        memory_map[MaCaco_OUT_s + slot + i] = memory_map[MaCaco_IN_s +
slot + i];
        memory_map[MaCaco_AUXIN_s + slot + i] =
memory_map[MaCaco_OUT_s + slot + i];
        memory_map[MaCaco_IN_s + slot + i] = Souliss_T1n_RstCmd;
    }

    memory_map[MaCaco_AUXIN_s + slot] = Souliss_T1n_Timed; // Set a timer for the state notification
    memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OnCoil; // Switch on the output
    memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd; // Reset
}
else

```

```

{      // There is no command

    if(memory_map[MaCaco_AUXIN_s + slot] > Souliss_T1n_Set)          //

Decrese the timer value
    memory_map[MaCaco_AUXIN_s + slot]--;
    else if(memory_map[MaCaco_AUXIN_s + slot] > 0)
// If we not getting new commands, the burst
{
    // is done, send the actual state
    memory_map[MaCaco_AUXIN_s + slot] = 0;
    i_trigger = Souliss_TRIGGERED;

}
}

// Update the trigger
if(i_trigger)
    *trigger = i_trigger;

return i_trigger;
}

//****************************************************************************
/*!
    Timer associated to T16
*/
//****************************************************************************
void Souliss_T16_Timer(U8 *memory_map, U8 slot)
{
    if(memory_map[MaCaco_IN_s + slot] > Souliss_T1n_Timed)          // Memory value is
used as timer
    {
        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_GoodNight;

        // Decrease timer and check the expiration
        if(--memory_map[MaCaco_IN_s + slot]) == Souliss_T1n_Timed)
            memory_map[MaCaco_IN_s + slot] = Souliss_T1n_OffCmd;

        // Bright down the light
        for(U8 i=1;i<4;i++)
            if(memory_map[MaCaco_OUT_s + slot + i])
                memory_map[MaCaco_OUT_s + slot + i]--;
            else
                memory_map[MaCaco_OUT_s + slot + i] = 0;

        // If there is no more light, declare it off
        if((memory_map[MaCaco_OUT_s + slot + 1] == Souliss_T1n_OffCoil) &&
(memory_map[MaCaco_OUT_s + slot + 2] == Souliss_T1n_OffCoil)

```

```

        && (memory_map[MaCaco_OUT_s + slot + 3] == Souliss_T1n_OffCoil))
    {
        memory_map[MaCaco_IN_s + slot] = Souliss_T1n_OffCmd;           // 
Reset
    }
}

//****************************************************************************
/*!
Define the use of Typical 18 : ON/OFF Digital Output with pulse output
with Timer Option
*/
//****************************************************************************
void Souliss_SetT18(U8 *memory_map, U8 slot)
{
    memory_map[MaCaco_TYP_s + slot] = Souliss_T18;
}

//****************************************************************************
/*!
Typical 18 : ON/OFF Digital Output

```

Handle one digital output based on hardware and software commands,  
output can be timed out.

This logic can be used for lights, wall socket and all the devices  
that has an ON/OFF behaviour. It has a pulsed output, and can be  
used with bistable relay or similar devices.

The actual state shall be reported with an external digital input  
such as an auxiliary contact or a current measure.

Hardware Command:

If using a monostable wall switch (press and spring return),  
each press will toggle the output status.

#define Souliss_T1n_ToggleCmd	0x01
-------------------------------	------

If using a bistable wall switch (press without return), the two  
switch position can be associated with the ON and OFF commands

#define Souliss_T1n_OnCmd	0x02
#define Souliss_T1n_OffCmd	0x04

The actual state shall be reported using these two feedbacks in  
relevant INPUT slot.

#define Souliss_T1n_OnFeedback	0x23
#define Souliss_T1n_OffFeedback	0x24

## Software Commands:

From any available software interface, these commands will turn the light ON and OFF.

#define Souliss_T1n_OnCmd	0x02
#define Souliss_T1n_OffCmd	0x04

### Command recap, using:

- 1(hex) as command, toggle the output
- 2(hex) as command, the output move to ON
- 4(hex) as command, the output move to OFF
- 0(hex) as command, no action

### Output status,

- 0(hex) for state OFF,
- 1(hex) for state ON,
- A1(hex) for pulsed output.

```
/*
 ****
U8 Souliss_Logic_T18(U8 *memory_map, U8 slot, U8 *trigger)
{
    // Look for input value, update output. If the output is not set, trig a data
    // change, otherwise just reset the input

    if ((memory_map[MaCaco_IN_s + slot] == Souliss_T1n_OffFeedback)) // If the pulse is
expired then
    {
        // set the State Feedback as OFF
        if(memory_map[MaCaco_AUXIN_s + slot] != Souliss_T1n_OffFeedback)
            *trigger = Souliss_TRIGGERED;

        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OffFeedback;           //
State Feedback as OFF
        memory_map[MaCaco_AUXIN_s + slot] = Souliss_T1n_OffFeedback; // Store the
actual State Feedback
        memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;             //
Reset
    }
    else if ((memory_map[MaCaco_IN_s + slot] == Souliss_T1n_OnFeedback)) // If the
pulse is expired then
    {
        // set the State Feedback as OFF
        if(memory_map[MaCaco_AUXIN_s + slot] != Souliss_T1n_OnFeedback)
            *trigger = Souliss_TRIGGERED;

        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OnFeedback;           //
State Feedback as OFF
        memory_map[MaCaco_AUXIN_s + slot] = Souliss_T1n_OnFeedback; // Store the
actual State Feedback
        memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;             //
    }
}
```

```

Reset
    }
    else if ((memory_map[MaCaco_IN_s + slot] == Souliss_T1n_ToggleCmd))

    {
        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_PulseCoil; // Change
output state
    }
    else if ((memory_map[MaCaco_IN_s + slot] == Souliss_T1n_OnCmd))
    {
        if (memory_map[MaCaco_AUXIN_s + slot] == Souliss_T1n_OffFeedback)
            memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_PulseCoil; // Change
output state if previously at OFF
    }
    else if ((memory_map[MaCaco_IN_s + slot] == Souliss_T1n_OffCmd))
    {
        if (memory_map[MaCaco_AUXIN_s + slot] == Souliss_T1n_OnFeedback)
            memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_PulseCoil; // Change
output state if previously at ON
    }
    else if (memory_map[MaCaco_OUT_s + slot] > Souliss_T1n_ResetCoil)
        memory_map[MaCaco_OUT_s + slot]--;
return *trigger;
}

//****************************************************************************
/*!
 Define the use of Typical 19 : RGB LED Strip
*/
//****************************************************************************
void Souliss_SetT19(U8 *memory_map, U8 slot)
{
    memory_map[MaCaco_TYP_s + slot] = Souliss_T19;
    memory_map[MaCaco_TYP_s + slot + 1] = Souliss_TRL;
}

//****************************************************************************
/*! Typical 19 : Single Color LED Strip

```

Handle one single color LED strip based on software commands, with automatic fade effects and "Good Night" feature. It use two (2) SLOT.

#### Hardware Command:

If using a monostable wall switch (press and spring return), each press will toggle the output status.

#define Souliss_T1n_ToggleCmd	0x01
#define Souliss_T1n_BrightUp	0x10
#define Souliss_T1n_BrightDown	0x20

If using a bistable wall switch (press without return), the two switch position can be associated with the ON and OFF commands

#define Souliss_T1n_OnCmd	0x02
#define Souliss_T1n_OffCmd	0x03

#### Software Commands:

From any available software interface, these commands will turn the light ON and OFF.

#define Souliss_T1n_OnCmd	0x02
#define Souliss_T1n_OffCmd	0x03
#define Souliss_T1n_BrightUp	0x10
#define Souliss_T1n_BrightDown	0x20
#define Souliss_T1n_Flash	0x21

Intensity can be set using the following command, followed by the value to be forced

#define Souliss_T1n_Set	0x22
-------------------------	------

#### Good Night:

The light goes off with a long time fade effect, this feature needs the external timer. Good Night mode is activated using if the input command is greater than

#define Souliss_T1n_Timed	0x30
---------------------------	------

A Good Night command can be sent via software or hardware commands, in case of hardware commands is usually associated to a long press of a monostable wall switch.

If INPUTVAL is the input value the output will be timed for nCYCLES of the associated timer.

nCYCLES = INPUTVAL - Souliss_T1n_Timed
--

#### Command recap, using:

- 1(hex) as command, toggle the output
- 2(hex) as command, the output move to ON
- 4(hex) as command, the output move to OFF
- 10(hex) as command, the light bright is increased
- 20(hex) as command, the light bright is decreased
- 21(hex) as command, the light flash on and off at each cycle
- 22(hex) as command, set the intensity
- >30(hex) as command, time the output to ON
- 0(hex) as command, no action

Output status,  
 - 0(hex) for output OFF,  
 - 1(hex) for output ON.

The next three slots contains the R, G and B values applied to the light.

```
/*
*****
U8 Souliss_Logic_T19(U8 *memory_map, U8 slot, U8 *trigger)
{
  U8 i_trigger=0;
          // Internal trigger

  // Look for input value, update output. If the output is not set, trig a data
  // change, otherwise just reset the input

  if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_ToggleCmd)           // Toggle
Command
  {
    // Toggle the actual status of the light
    if(memory_map[MaCaco_OUT_s + slot] == Souliss_T1n_OffCoil)
      memory_map[MaCaco_IN_s + slot] = Souliss_T1n_OnCmd;

    else if(memory_map[MaCaco_OUT_s + slot] == Souliss_T1n_OnCoil ||
memory_map[MaCaco_OUT_s + slot] == Souliss_T1n_GoodNight )
      memory_map[MaCaco_IN_s + slot] = Souliss_T1n_OffCmd;
    else
      memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;
  }
  else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_OffCmd)           // Off
Command
  {
    // Trigger the change and save the actual color
    if(memory_map[MaCaco_OUT_s + slot] != Souliss_T1n_OffCoil)
    {
      memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OffCoil;                //
Switch off the light state
      i_trigger = Souliss_TRIGGERED;
      // Trig the change
    }

    // Fade out and turn off the light step wise
    if(memory_map[MaCaco_OUT_s + slot + 1])
      memory_map[MaCaco_OUT_s + slot + 1]--;
  }

  // Once is off, reset
  if((memory_map[MaCaco_OUT_s + slot + 1] == 0))
    memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;                   //
}
```

Reset

```

    }
    else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_OnCmd)
    {
        if(memory_map[MaCaco_OUT_s + slot] != Souliss_T1n_OnCoil)
            i_trigger = Souliss_TRIGGERED;

        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OnCoil;           //
```

Switch on the output

```

// If there were no color set, use a light white
if((memory_map[MaCaco_AUXIN_s + slot + 1] == 0))
{
    memory_map[MaCaco_OUT_s + slot + 1] = 0xAA;

// Set a light white
    memory_map[MaCaco_AUXIN_s + slot + 1] =
memory_map[MaCaco_OUT_s + slot + 1];

    memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;

// Reset
}
else
{
    // Fade in to the last color
    if(memory_map[MaCaco_OUT_s + slot + 1] <
(memory_map[MaCaco_AUXIN_s + slot + 1]))
        memory_map[MaCaco_OUT_s + slot + 1]++;
}

// Once is on, reset
if((memory_map[MaCaco_OUT_s + slot + 1] == memory_map[MaCaco_AUXIN_s
+ slot + 1]))
    memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;

// Reset
}
else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_BrightUp)          // Increase
the light value
{
    // Increase the light value
    if(memory_map[MaCaco_OUT_s + slot + 1] < 255 - Souliss_T1n_BrightValue)
        memory_map[MaCaco_OUT_s + slot + 1] += Souliss_T1n_BrightValue;

    memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;                   //

Reset
}
else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_BrightDown)
    // Decrease the light value
{
    // Decrease the light value
```

```

        if(memory_map[MaCaco_OUT_s + slot + 1] > Souliss_T1n_BrightValue)
            memory_map[MaCaco_OUT_s + slot + 1] -= Souliss_T1n_BrightValue;

            memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd; //Reset
        }

        else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_Flash)
            // Turn ON and OFF at each cycle
        {
            // If the light was on
            if(memory_map[MaCaco_OUT_s + slot] == Souliss_T1n_OnCoil)
            {
                // Switch ON and OFF until a new command is received
                if((memory_map[MaCaco_OUT_s + slot + 1] ==
memory_map[MaCaco_AUXIN_s + slot + 1]))
                    memory_map[MaCaco_OUT_s + slot + 1] = 0;
                    // Set output to zero
                else if((memory_map[MaCaco_OUT_s + slot + 1] == 0))
                    memory_map[MaCaco_OUT_s + slot + 1] =
memory_map[MaCaco_AUXIN_s + slot + 1]; // Set output to previous value
                }
            }

            else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_Set)
            {
                // Set the new color
                memory_map[MaCaco_OUT_s + slot + 1] = memory_map[MaCaco_IN_s + slot +
1];
                memory_map[MaCaco_AUXIN_s + slot + 1] = memory_map[MaCaco_OUT_s +
slot + 1];
                memory_map[MaCaco_IN_s + slot + 1] = Souliss_T1n_RstCmd;

                memory_map[MaCaco_AUXIN_s + slot] = Souliss_T1n_Timed; //Set a timer for the state notification
                memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OnCoil; //Switch on the output
                memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd; //Reset
            }

            else
            {
                // There is no command
                if(memory_map[MaCaco_AUXIN_s + slot] > Souliss_T1n_Set) //Decrese the timer value
                    memory_map[MaCaco_AUXIN_s + slot]--;
                else if(memory_map[MaCaco_AUXIN_s + slot] > 0)
                    // If we not getting new commands, the burst
                    {
                        // is done, send the actual state
                        memory_map[MaCaco_AUXIN_s + slot] = 0;
                    }
            }
        }
    }
}

```

```

        i_trigger = Souliss_TRIGGERED;

    }

}

// Update the trigger
if(i_trigger)
    *trigger = i_trigger;

return i_trigger;
}

//****************************************************************************
/*!
    Timer associated to T19
*/
//****************************************************************************
void Souliss_T19_Timer(U8 *memory_map, U8 slot)
{
    if(memory_map[MaCaco_IN_s + slot] > Souliss_T1n_Timed)           // Memory value is
used as timer
    {
        memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_GoodNight;

        // Decrease timer and check the expiration
        if(--memory_map[MaCaco_IN_s + slot]) == Souliss_T1n_Timed)
            memory_map[MaCaco_IN_s + slot] = Souliss_T1n_OffCmd;

        // Bright down the light
        if(memory_map[MaCaco_OUT_s + slot + 1])
            memory_map[MaCaco_OUT_s + slot + 1]--;
        else
            memory_map[MaCaco_OUT_s + slot + 1] = 0;

        // If there is no more light, declare it off
        if((memory_map[MaCaco_OUT_s + slot + 1] == Souliss_T1n_OffCoil))
            memory_map[MaCaco_IN_s + slot] = Souliss_T1n_OffCmd;           //

Reset
    }
}

//****************************************************************************
/*!
    Define the use of Typical 1A : Digital pass through
*/
//****************************************************************************
void Souliss_SetT1A(U8 *memory_map, U8 slot)
{

```

```

        memory_map[MaCaco_TYP_s + slot] = Souliss_T1A;
}

```

```
*****  
/*!
```

### Typical 1A : Digital Input Pass Through

It read read data from input and pass them through to the output

Hardware Command:

There aren't command, but every value is copied in output

Command recap, using:

- Any value

Output status,

- Any value

```
*/  
*****  
U8 Souliss_Logic_T1A(U8 *memory_map, U8 slot, U8 *trigger)  
{  
    U8 i_trigger=0;  
    // Internal trigger  
  
    // Look for input value, update output. If the output is not set, trig a data  
    // change, otherwise just reset the input  
  
    if(memory_map[MaCaco_IN_s + slot] != memory_map[MaCaco_OUT_s + slot])  
    {  
        memory_map[MaCaco_OUT_s + slot] = memory_map[MaCaco_IN_s +  
slot];  
        i_trigger = Souliss_TRIGGERED;  
        // Trig change  
    }  
    else  
    {  
        memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd; //  
Reset  
    }  
  
    // Update the trigger  
    if(i_trigger)  
        *trigger = i_trigger;  
  
    return i_trigger;  
}  
*****
```

```
/*
 * Define the use of Typical 1B : Position Constrained ON/OFF Digital Output
 */
void Souliss_SetT1B(U8 *memory_map, U8 slot)
{
    memory_map[MaCaco_TYP_s + slot] = Souliss_T1B;
}

/*
 * Typical 1B : Position Constrained ON/OFF Digital Output

```

Handle one digital output based on position of the mobile user interface, that is requested to push continuously data in order to keep the output in the ON condition.

The output goes to OFF once the user interface stop to push data.

Is expected that the user interface push data only via WiFi, this give a position constrained behaviour.

#### Software Commands:

From any available software interface, these commands will turn the light ON and OFF.

#define Souliss_T1n_OnCmd	0x02
#define Souliss_T1n_OffCmd	0x04
#define Souliss_T1n_PositionOnCmd	0x31

The Souliss\_T1n\_PositionOnCmd force the output to Souliss\_T1n\_OnCoil and start a count-down timer. The timer got a reset every time that the user interface send a the command Souliss\_T1n\_PositionOnCmd.

If the user interface send periodically the Souliss\_T1n\_PositionOnCmd the output stays in Souliss\_T1n\_OnCoil, and goes back to Souliss\_T1n\_OffCoil only when the user interface is out of WiFi range and is no longer able to send data.

#### Output status,

- 0(hex) for output OFF,
- 1(hex) for output ON.

```
/*
 */
U8 Souliss_Logic_T1B(U8 *memory_map, U8 slot, U8 *trigger)
{
    U8 i_trigger=0;
        // Internal trigger

```

```

// Look for input value, update output. If the output is not set, trig a data
// change, otherwise just reset the input

if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_OffCmd)
{
    if(memory_map[MaCaco_OUT_s + slot] != Souliss_T1n_OffCoil)
        i_trigger = Souliss_TRIGGERED;

    memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OffCoil; // Switch off the output
    memory_map[MaCaco_AUXIN_s + slot] = 0;
    // Reset the timer
    memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd; // Reset
}

else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_OnCmd)
{
    if(memory_map[MaCaco_OUT_s + slot] != Souliss_T1n_OnCoil)
        i_trigger = Souliss_TRIGGERED;

    memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OnCoil; // Switch on the output
    memory_map[MaCaco_AUXIN_s + slot] = 0;
    // Reset the timer
    memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd; // Reset
}

else if (memory_map[MaCaco_IN_s + slot] == Souliss_T1n_PositionOnCmd)
{
    if(memory_map[MaCaco_OUT_s + slot] != Souliss_T1n_OnCoil)
        i_trigger = Souliss_TRIGGERED;

    memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OnCoil; // Switch on the output
    memory_map[MaCaco_AUXIN_s + slot] = Souliss_T1n_Timed_StdVal; // Set the timer
    memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd; // Reset
}

// Count-down and switch OFF the output if the timer is expired
if(memory_map[MaCaco_AUXIN_s + slot] > Souliss_T1n_Timed)
    memory_map[MaCaco_AUXIN_s + slot]--;
else if(memory_map[MaCaco_AUXIN_s + slot] != 0)
{
    if(memory_map[MaCaco_OUT_s + slot] != Souliss_T1n_OffCoil)
        i_trigger = Souliss_TRIGGERED;
}

```

```
    memory_map[MaCaco_OUT_s + slot] = Souliss_T1n_OffCoil;          //  
Switch off the output  
    memory_map[MaCaco_AUXIN_s + slot] = 0;  
    // Reset the timer  
    memory_map[MaCaco_IN_s + slot] = Souliss_T1n_RstCmd;          //  
Reset  
}  
  
// Update the trigger  
if(i_trigger)  
    *trigger = i_trigger;  
  
return i_trigger;  
}
```

## T2N.CPP

```
*****
```

Souliss  
Copyright (C) 2011 Veseo

This program 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 3 of the License, or (at your option) any later version.

This program 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 program. If not, see <<http://www.gnu.org/licenses/>>.

Originally developed by Dario Di Maio and Alessandro DelPex

```
*****  
/*!  
 \file  
 \ingroup  
  
*/  
*****  
/*!  
 Define the use of Typical 21 : Motorized devices with limit switches  
*/  
*****  
void Souliss_SetT21(U8 *memory_map, U8 slot)  
{  
     memory_map[MaCaco_TYP_s + slot] = Souliss_T21;  
}  
*****  
/*!  
 Typical 21 : Motorized devices with limit switches
```

It handle OPEN / CLOSE devices with alternate direction moving through the STOP state. Is used for garage doors or generally devices that need to be fully opened or closed.

Once a command, it remain active until the relevant limit switch is recognized or the associated timer is expired. Timer must be used for proper release of command.

Limit switches are not mandatory, but if not used, the device engine

shall be protected from extra current or shall have its own control center.

#### Hardware Limit Switches :

Using bistable switches for identification of open and close position

```
#define Souliss_T2n_LimSwitch_Close      0x08
#define Souliss_T2n_LimSwitch_Open        0x10
```

#### Hardware Command:

Using a monostable wall switch (press and spring return) or a software command from user interface, each press will toggle the output status.

```
#define Souliss_T2n_ToggleCmd          0x04
```

#### Software Command:

```
#define Souliss_T2n_CloseCmd          0x01
#define Souliss_T2n_OpenCmd            0x02
#define Souliss_T2n_StopCmd           0x03
```

#### Output status:

- 1(hex) for CLOSE,
- 2(hex) for OPEN,
- 3(hex) for STOP.

```
*/
 ****
U8 Souliss_Logic_T21(U8 *memory_map, U8 slot, U8 *trigger, U8
timeout=Souliss_T2n_Timer_Val)
{
    U8 i_trigger=0;
                // Internal trigger
    if(timeout<=Souliss_T2n_Timer_Off)      timeout=Souliss_T2n_Timer_Val;

    // Look for input value, update output. If the output is not set, trig a data
    // change, otherwise just reset the input

    if((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_ToggleCmd) ||
(memory_map[MaCaco_IN_s + slot] == Souliss_T2n_StopCmd) || (memory_map[MaCaco_IN_s +
slot] == Souliss_T2n_OpenCmd_Local) || (memory_map[MaCaco_IN_s + slot] ==
Souliss_T2n_CloseCmd_Local))
    {
        if(memory_map[MaCaco_IN_s + slot] == Souliss_T2n_ToggleCmd)
        {
            // Change the output value, between OPEN and CLOSE always OFF is
performed
            if((memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_Coil_Close) ||
```

```

(memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_Coil_Open))
{
    memory_map[MaCaco_AUXIN_s + slot] =
memory_map[MaCaco_OUT_s + slot]; // Save actual state
    memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Stop;
// Off Command
}
else if((memory_map[MaCaco_AUXIN_s + slot] ==
Souliss_T2n_Coil_Close) || (memory_map[MaCaco_OUT_s + slot] ==
Souliss_T2n_LimSwitch_Close))
    memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Open;
// Open Command
else if((memory_map[MaCaco_AUXIN_s + slot] ==
Souliss_T2n_Coil_Open) || (memory_map[MaCaco_OUT_s + slot] ==
Souliss_T2n_LimSwitch_Open))
    memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Close;
// Close command
else
    memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Open;
// If state is undefined, Open Command
}
else if((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_OpenCmd_Local) &&
((memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_Coil_Stop) ||
(memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_LimSwitch_Close)))
    memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Open;
// Open Command
else if((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_CloseCmd_Local) &&
((memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_Coil_Stop) ||
(memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_LimSwitch_Open)))
    memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Close;
// Close command
else
    memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Stop;
// Stop Command

// If a command was issued, set the timer
if((memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_Coil_Open) ||
(memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_Coil_Close))
    memory_map[MaCaco_AUXIN_s + slot] = timeout;
// Set timer value

memory_map[MaCaco_IN_s + slot] = Souliss_T2n_RstCmd;
// Reset
i_trigger = Souliss_TRIGGERED;
}
else if((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_LimSwitch_Close) ||
((memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_Coil_Close) &&
(memory_map[MaCaco_AUXIN_s + slot] == Souliss_T2n_Timer_Off)))
{

```

```

        memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_LimSwitch_Close;
        // Close Limit Switch
        memory_map[MaCaco_IN_s + slot] = Souliss_T2n_RstCmd;
        // Reset
        i_trigger = Souliss_TRIGGERED;
    }
    else if((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_LimSwitch_Open) ||
((memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_Coil_Open) &&
(memory_map[MaCaco_AUXIN_s + slot] == Souliss_T2n_Timer_Off)))
    {
        memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_LimSwitch_Open;
        // Open Limit Switch
        memory_map[MaCaco_IN_s + slot] = Souliss_T2n_RstCmd;
        // Reset
        i_trigger = Souliss_TRIGGERED;
    }

    // Update the trigger
    if(i_trigger)
        *trigger = i_trigger;

    return i_trigger;
}

//****************************************************************************
/*!
    Timer associated to T21, timeout the OPEN/CLOSE commands
*/
//****************************************************************************
void Souliss_T21_Timer(U8 *memory_map, U8 slot)
{
    if(memory_map[MaCaco_AUXIN_s + slot] > Souliss_T2n_Timer_Off)
        // Memory value is used as timer
        memory_map[MaCaco_AUXIN_s + slot]--;                                // Decrease timer
}

//****************************************************************************
/*!
    Define the use of Typical 22 : Motorized devices with limit switches
*/
//****************************************************************************
void Souliss_SetT22(U8 *memory_map, U8 slot)
{
    memory_map[MaCaco_TYP_s + slot] = Souliss_T22;
    memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Stop;
}

//****************************************************************************

```

/\*!

Typical 22 : Motorized devices with limit switches and middle position

It handle OPEN / CLOSE devices with alternate direction, dedicated commands for OPEN, CLOSE and STOP are available. Every state change goes through the STOP command for security reason.  
Is used for curtains or other devices that need an adjustable position.

Once a command, it remain active until the relevant limit switch is recognized or the associated timer is expired. Timer must be used for proper release of command.

Limit switches are not mandatory, but if not used, the device engine shall be protected from extra current or shall have its own control center.

Hardware Limit Switches :

Using bistable switches for identification of open and close position

```
#define Souliss_T2n_LimSwitch_Close      0x08
#define Souliss_T2n_LimSwitch_Open        0x10
```

Hardware and/or Software Command:

Using a monostable wall switch (press and spring return) or a software command from user interface, each press will toggle the output status.

```
#define Souliss_T2n_CloseCmd_Local    0x08
#define Souliss_T2n_OpenCmd_Local       0x10
#define Souliss_T2n_StopCmd
```

0x04

Following constant are defined for sketch source compatibility with versions

&lt; A6.1.1

and their use is now deprecated.

```
#define Souliss_T2n_CloseCmd          0x01
#define Souliss_T2n_OpenCmd            0x02
```

This commands are designed to be used by an application (Souliss App, OpenHAB binding, user applications)

in order to support software "scenario". When the Open/Close command is received it is always executed;

if the motor is running opposite direction it stops for 4 cycles then it revert motion.

```
#define Souliss_T2n_CloseCmd_SW      0x01
#define Souliss_T2n_OpenCmd_SW         0x02
#define Souliss_T2n_StopCmd
```

0x03

Command recap, using:

- 0x01(hex) as command, Software CLOSE request (stop 4 cycles if opening)
- 0x02(hex) as command, Software OPEN request (stop 4 cycles if closing)
- 0x04(hex) as command, STOP request
- 0x08(hex) as command, CLOSE request (stop if opening)
- 0x10(hex) as command, OPEN request (stop if closing)

Output status:

- 1(hex) for CLOSE,
- 2(hex) for OPEN,
- 3(hex) for STOP.

```
/*
*****
U8 Souliss_Logic_T22(U8 *memory_map, U8 slot, U8 *trigger, U8
timeout=Souliss_T2n_Timer_Val)
{
    U8 i_trigger=0;
                // Internal trigger
    if(timeout<=Souliss_T2n_Timer_Off)      timeout=Souliss_T2n_Timer_Val;
    else if (timeout>Souliss_T2n_Timer_Val) timeout=Souliss_T2n_Timer_Val;

    // Look for input value, update output. If the output is not set, trig a data
    // change, otherwise just reset the input

    if((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_CloseCmd_SW) ||
       (memory_map[MaCaco_IN_s + slot] == Souliss_T2n_OpenCmd_SW) ||
       (memory_map[MaCaco_IN_s + slot] == Souliss_T2n_OpenCmd_Local) ||
       (memory_map[MaCaco_IN_s + slot] == Souliss_T2n_CloseCmd_Local) ||
       (memory_map[MaCaco_IN_s + slot] == Souliss_T2n_StopCmd))

    {
        // Change the output value, between OPEN and CLOSE always STOP is performed

        if((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_StopCmd))
            memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Stop;
        // Stop Command
        else if(((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_OpenCmd_Local)) &&
&& (((memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_Coil_Stop) ||
&& !Souliss_T2n_IsTemporaryStop) || (memory_map[MaCaco_OUT_s + slot] ==
Souliss_T2n_LimSwitch_Close) || (memory_map[MaCaco_OUT_s + slot] ==
Souliss_T2n_NoLimSwitch)))
            memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Open;
        // Open Command
        else if(((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_CloseCmd_Local)) &&
&& (((memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_Coil_Stop) ||
&& !Souliss_T2n_IsTemporaryStop) || (memory_map[MaCaco_OUT_s + slot] ==
```

```

Souliss_T2n_LimSwitch_Open) || (memory_map[MaCaco_OUT_s + slot] ==
Souliss_T2n_NoLimSwitch)))
    memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Close;
    // Close command
    else if((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_OpenCmd_Local) ||
(memory_map[MaCaco_IN_s + slot] == Souliss_T2n_CloseCmd_Local))
        memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Stop;
    // Stop Command
    else if((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_OpenCmd_SW) &&
        (((memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_Coil_Stop) ||
&& !Souliss_T2n_IsTemporaryStop) || (memory_map[MaCaco_OUT_s + slot] ==
Souliss_T2n_LimSwitch_Close) || (memory_map[MaCaco_OUT_s + slot] ==
Souliss_T2n_NoLimSwitch)))
        memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Open;
        // Open SW Command immediately executable because motor isn't running
        else if((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_OpenCmd_SW) &&
(memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_Coil_Close))
    {
        // Open SW Command that can't
be executed because motor is running opposite direction
        memory_map[MaCaco_AUXIN_s + slot] = Souliss_T2n_TimedStop_Val;
        // Set timer value for the temporary Stop state
        memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Stop;      //
Temporary stop
    }
    else if((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_OpenCmd_SW) &&
(memory_map[MaCaco_AUXIN_s + slot] == Souliss_T2n_TimedStop_Off))
        memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Open;
        // Open SW Command executable because temporary stop is over
        else if((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_CloseCmd_SW) &&
            (((memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_Coil_Stop) ||
&& !Souliss_T2n_IsTemporaryStop) || (memory_map[MaCaco_OUT_s + slot] ==
Souliss_T2n_LimSwitch_Open) || (memory_map[MaCaco_OUT_s + slot] ==
Souliss_T2n_NoLimSwitch)))
            memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Close;
            // Close SW command      immediately executable because motor isn't running
            else if((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_CloseCmd_SW) &&
(memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_Coil_Open))
    {
        // Close SW Command that can't
be executed because motor is running opposite direction
        memory_map[MaCaco_AUXIN_s + slot] = Souliss_T2n_TimedStop_Val;
        // Set timer value for the temporary Stop state
        memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Stop;      //
Temporary stop
    }

```

```

else if((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_CloseCmd_SW) &&
(memory_map[MaCaco_AUXIN_s + slot] == Souliss_T2n_TimedStop_Off))
    memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_Coil_Close;
// Close SW Command executable because temporary stop is over

// If a command was issued, set the timer
if(!Souliss_T2n_IsTemporaryStop)
{
    memory_map[MaCaco_AUXIN_s + slot] = timeout;
    // Set timer value
    memory_map[MaCaco_IN_s + slot] = Souliss_T2n_RstCmd;
    // Reset command

    // Set the trigger
    i_trigger = Souliss_TRIGGERED;
}
}

else if((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_LimSwitch_Close) ||
        ((memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_Coil_Close) &&
         (memory_map[MaCaco_AUXIN_s + slot] ==
Souliss_T2n_Timer_Off)))
{
    memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_LimSwitch_Close;
    // Close Limit Switch
    memory_map[MaCaco_IN_s + slot] = Souliss_T2n_RstCmd;
    // Reset
    i_trigger = Souliss_TRIGGERED;
}
else if((memory_map[MaCaco_IN_s + slot] == Souliss_T2n_LimSwitch_Open) ||
        ((memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_Coil_Open) &&
         (memory_map[MaCaco_AUXIN_s + slot] ==
Souliss_T2n_Timer_Off)))
{
    memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_LimSwitch_Open;
    // Open Limit Switch
    memory_map[MaCaco_IN_s + slot] = Souliss_T2n_RstCmd;
    // Reset
    i_trigger = Souliss_TRIGGERED;
}
else if((memory_map[MaCaco_OUT_s + slot] == Souliss_T2n_Coil_Stop) &&
        (memory_map[MaCaco_AUXIN_s + slot] == Souliss_T2n_Timer_Off))
{
    memory_map[MaCaco_OUT_s + slot] = Souliss_T2n_NoLimSwitch;
    // No Limit Switch
    memory_map[MaCaco_IN_s + slot] = Souliss_T2n_RstCmd;
    // Reset
    i_trigger = Souliss_TRIGGERED;
}

```

```
// Update the trigger
if(i_trigger)
    *trigger = i_trigger;

return i_trigger;
}

//****************************************************************************
/*!
    Timer associated to T22, timeout the OPEN/CLOSE commands
*/
//****************************************************************************
void Souliss_T22_Timer(U8 *memory_map, U8 slot)
{
    // Memory value is used as timer
    if(((memory_map[MaCaco_AUXIN_s + slot] > Souliss_T2n_Timer_Off) &&
        (memory_map[MaCaco_AUXIN_s + slot] <= Souliss_T2n_Timer_Val)) ||
        Souliss_T2n_IsTemporaryStop )
    {
        memory_map[MaCaco_AUXIN_s + slot]--;
        // Decrease timer
    }
}
```