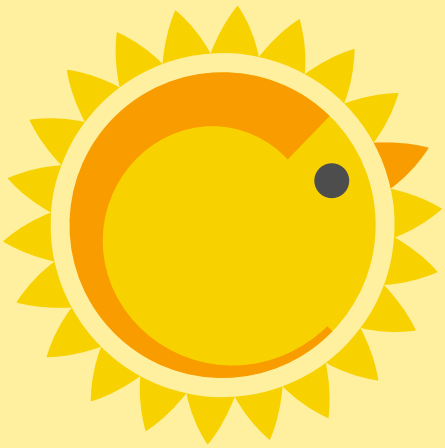


pio
programmable
input_output
interface

PIO LIBRARY

FUNCTION REFERENCE MANUAL

Intellisys S.r.l.
info@intellisys.it | www.intellisys.it

General Index

INTRODUCTION

Aim

PIO is a **P**rogrammable peripheral of **I**nputs and **O**utputs, which we called **PIO interface**.
This manual describes the method of configuring a PIO interface i.e. the way the interface can be set up to perform some control functions with the use of programming languages, like **C**, **JAVA** and **PYTHON**.

Hardware

The PIO interface is a microprocessor based configurable hardware, **installed on RASPBERRY PI's module**, which is applicable to a lot of buildings and plants to perform any type of applications on field. PIO interface has 10 points of I/O (Inputs and Outputs) with which you can acquire information, analogue and/or digital from sensors on field, and you can control motors, actuators, pumps, valves and others devices on field.

Analogue Inputs

The analogue Inputs are used to make measurements of temperature, pressure, position, level, gas presence (for example CO2 in the air), humidity etc. The measurements are made by fitting the appropriate sensor which produces a continuous electrical signal that is proportional to the measurement being made.

Digital Inputs

The digital Inputs are used to monitor the status of the plant and/or switches. This status will be either open or close (OFF or ON).

Universal Inputs

The term "Universal Input" is used to define an input channel that may be configured to be either analogue (Voltage, Current, NTC 10K) or digital.

Analogue Outputs

The analogue Outputs are used to control valve's position, dampers, generic actuators etc, however where a continuous voltage is required. They are also used to drive multiple relay modules where the relay selected depends from voltage level given to the output.

Digital outputs

The Digital Outputs are used to switch commands ON or OFF to the plant.

Led

PIO has the possibility to control 3 led, 1 red and 2 green.

I/O numbers and details

1. Input Channels
   - **4** opto-isolated digital inputs named **DI1**, **DI2**, **DI3**, **DI4.**
   - **2** universal inputs, named **UI1**, **UI2**, configurables in VOLTAGE, CURRENT 0..20 mA and 4..20 mA, NTC 10K passive sensor for temperature in the range -30 °C .. +110 °C, DIGITAL.
2. Output Channels
   - **2** digital outputs with relay named **DO1**, **DO2**.
   - **2** analogue outputs 0..10 Vcc named **AO1**, **AO2**.

For more information please see PIO's data-sheet.

Parameters

All the parameters definite in the PIO LIBRARY are stored in a volatile memory and they must be initialized, after each power on, in the INIT procedure. If you want to store them in a file you must write your own code program.

SYSTEM LIBRARY : PIO_SYSTEM

- `int Pio_SYSTEM_Init(void);`
- `int Pio_SYSTEM_Dispose(void);`

SYSTEM : General notes

SYSTEM's library is a set of software functions that you must use to initialize PIO hardware interface. PIO has some hardware devices, like communication's bus, that must be initialized before the library function can be called with your program code.

- How to do to use library function - :

- No hardware settings to do with your hands on the PIO interface;
- enable the PIO hardware device calling the function `Pio_SYSTEM_Init()`. **This is the most important thing to do using your program code**. If you don't, the PIO interface will not be accessible and the other library functions will be out of order.

Pio_SYSTEM_Init

Declaration `int Pio_SYSTEM_Init(void)`

Description Initializes PIO interface to permit accessibility to library functions. Normally the function `Pio_SYSTEM_Init` is called inside of the your INIT procedure.

Parameter Type/Description
none none

Return value integer number - success, warning or error code.

Related functions **none.**

Platform RASPBIAN on RASPBERRY PI

Return value

+1 for SUCCESS : the PIO interface is been successful initialized.
0 for warning : the Pio interface has already been initialized.
-1 for error : the PIO interface has not been correctly initialized.

Example codes

Ex.1 : Initialize PIO interface before writing your program code to use the library functions. This is the most important setting to do.

```
int main (void)
{
    Pio_SYSTEM_Init();

    while (1)
    {
        // YOUR CODE: read sensor's value, control analogue or digital outputs,
        // blink a led etc.
    }
}
```

Ex.2 : Initialize PIO interface before writing your program code to use the library functions. This is the most important setting to do.

```
int main (void)
{
    int system_err;

    system_err  = Pio_SYSTEM_Init();
    printf ("The system error number is : %i\n", system_err);

    while (1)
    {
        // YOUR CODE: read sensor's value, control analogue or digital outputs,
        // blink a led etc.
    }
}
```

Ex.3 : Initialize PIO interface before writing your program code to use the library functions. This is the most important setting to do.

```
void Init(void)
{
    Pio_SYSTEM_Init();
    // Write your program code of initialization, for example to initialize
    // the Real Time Clock library (see section RTC ahead on this document).
}

int main (void)
{
    Init();

    while (1)
    {
        // YOUR CODE: read sensor's value, control analogue or digital outputs,
        // blink a led etc.
    }
}
```

Pio_SYSTEM_Dispose

Declaration `int Pio_SYSTEM_Dispose(void)`

Description De-allocates all the resources previously setted with calling to the `Pio_SYSTEM_Init` function. Normally the function `Pio_SYSTEM_Dispose` is called when you want put the Pio interface in a Stop mode. After that `Pio_SYSTEM_Dispose` is called, all the inputs and outputs of the Pio interface will be out of service.
To restart the Pio interface you must call the Pio_SYSTEM_Init function again.

Parameter	Type/Description
none	none

Return value integer number – success or error code.

Related functions **none.**

Platform RASPBIAN on RASPBERRY PI

Return value

+1 for SUCCESS : the PIO interface library is been successful released.
-1 for error : the PIO interface has not been correctly released.

Example codes

Ex.1 : Initializes PIO interface, increments a variable every 1 second and when its value is equal 10 then releases the Pio interface.

```
#include <unistd.h>

int main (void)
{
    int i;
    Pio_SYSTEM_Init();
```

```
    while (1)
    {
        delay (1000);
        i++;
        if (i >= 10) {Pio_SYSTEM_Dispose();}
    }
}
```

UNIVERSAL INPUT LIBRARY : PIO_UI

- `int Pio_UI_Type_Write (int id, int value);`
- `int Pio_UI_Type_Read (int id);`
- `int Pio_UI_BottomVolt_Write (int id, int value);`
- `int Pio_UI_BottomVolt_Read (int id);`
- `int Pio_UI_TopVolt_Write (int id, int value);`
- `int Pio_UI_TopVolt_Read (int id);`
- `int Pio_UI_BottomRange_Write (int id, float value);`
- `float Pio_UI_BottomRange_Read (int id);`
- `int Pio_UI_TopRange_Write (int id, float value);`
- `float Pio_UI_TopRange_Read (int id);`
- `float Pio_UI_Value_Read (int id);`

UI : General notes

The library of the Universal Inputs (UI) is a set of software functions with which to manage sensors, physically connected to the terminal of the PIO interface. The library scales an analogue input reading into engineering units. Analogue input signals are generated by a sensor, a physical device that converts a physical property (e.g. temperature, pressure, humidity, gas concentration etc) into an electrical signal. The analogue electrical signals generated by sensors can be: a current in mA, or voltage in Volt of any size. The PIO's inputs accept current in two ranges, the first from 0 to 20 mA and the second from 4 to 20 mA.
As regards instead the voltage signals, the PIO's inputs accept a voltage in any values inside the range 0..10 V. The PIO interface accept NTC 10K signals from thermistor. The universal input channels are used by both analogue sensor and digital input.

Each universal input has five possible types of configuration, enumerated from 0 to 4.

- Case 0 - **VOLTAGE** : any size inside 0..10 V range (factory default).
- Case 1 - **CURRENT 4..20 mA** : the input can change electrical signal from 4 mA to 20 mA.
- Case 2 - **CURRENT 0..20 mA** : the input can change electrical signal from 0 mA to 20 mA.
- Case 3 - **THERMISTOR NTC 10K** : the input is a thermistor NTC 10K (10000 Ω at 25 °C) and the range of measurement is -20..+110 °C. See data-sheet for sensor characteristics.
- Case 4 - **DIGITAL** : the input is ON/OFF (close/open contact).

To use correctly a universal input reading you need to do two activities. The first is a easy hardware setting on the PIO interface, the second is a software setting utilizing the PIO_UI library.

The hardware setting is very easy to do because you must put a jumper corresponding to its input, in one of 3 possible positions:

- NO jumper to have VOLTAGE input,
- in the first position for CURRENT input,
- in the second position for NTC and DIGITAL input.

Please see the PIO data-sheet for more details.

To understand how to use the PIO_UI library you need to familiarize with 4 parameters that are:

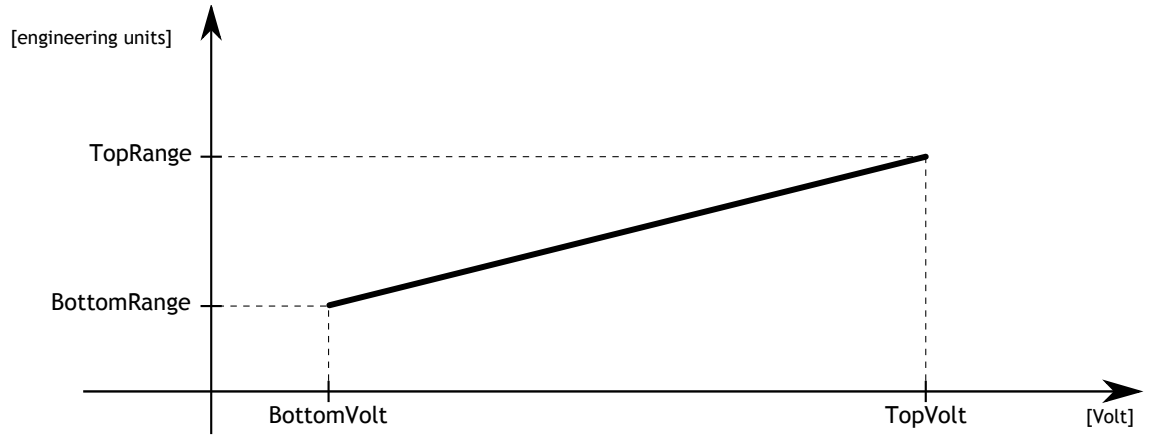
1. "BottomVolt",
2. "TopVolt",

3. “BottomRange”,
4. “TopRange”.

The parameters “BottomVolt” and “TopVolt” are used only when the input is reading VOLTAGE signal whilst “BottomRange” and “TopRange” are used for both VOLTAGE and CURRENT signals. The four parameters are not used for NTC 10K and DIGITAL input signals.

Voltage universal input

Consider the next figure.



If you must configure an universal input that receives from sensor a signal voltage from 0 V to 10 V, and the sensor must read temperature from 0 °C to 50 °C, the parameters are:
"BottomVolt" = 0, "BottomRange" = 0, "TopVolt" = 10 and "TopRange" = 50.

If you must configure an universal input that receives from servomotor a feedback position with voltage signal from 2 V and 10 V, and the universal input must read percentage of position from 0% to 100%, the parameters are:
"BottomVolt" = 2, "BottomRange" = 0, "TopVolt" = 10 and "TopRange" = 100.

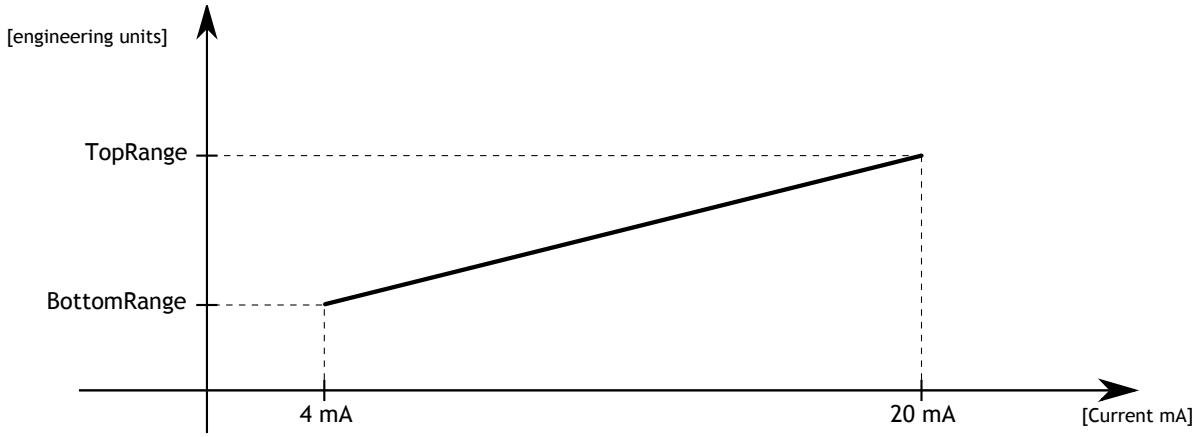
If a sensor reads the presence of CO2 in a space from 0 ppm to 2000 ppm and the electrical signal is from 4 V to 8 V the parameters are:
"BottomVolt" = 4, "BottomRange" = 0, "TopVolt" = 8 and "TopRange" = 2000.

- How to do to use library function - :

- Set the related jumper of the input on VOLTAGE mode. Please see the PIO's datasheet.
- The PIO interface has two universal inputs, enumerated from one to two (UI1, UI2). Passing id = 1 to the function means you are configuring the input number one. Passing id = 2 to the function means you are configuring the input number two.
- Set to zero (0) the number of the type of input, using `PIO_UI_Type_Write` function.
- Set the "BottomVolt" value using `PIO_UI_BottomVolt_Write` function.
- Set the "TopVolt" value using `PIO_UI_TopVolt_Write` function.
- Be sure "TopVolt" is greater than "BottomVolt".
- Set the "BottomRange" value, in engineering units, using `PIO_UI_BottomRange_Write` function.
- Set the "TopRange" value, in engineering units, using `PIO_UI_TopRange_Write` function.
- Read the value, in engineering units, from sensor connected to the input using `PIO_UI_Value_Read` function.

Current 4..20 mA universal input

Consider the next figure.



If you must configure an universal input that receive from sensor a signal from 4 mA to 20 mA and the sensor must read temperature from 0 °C to 50 °C the parameters are:
"BottomRange" = 0 and "TopRange" = 50.

If you must configure an universal input that receives from a servomotor a feedback position with 4..20 mA signal, and the universal input must read percentage of position from 0% to 100%, the parameters are:
"BottomRange" = 0 and "TopRange" = 100.

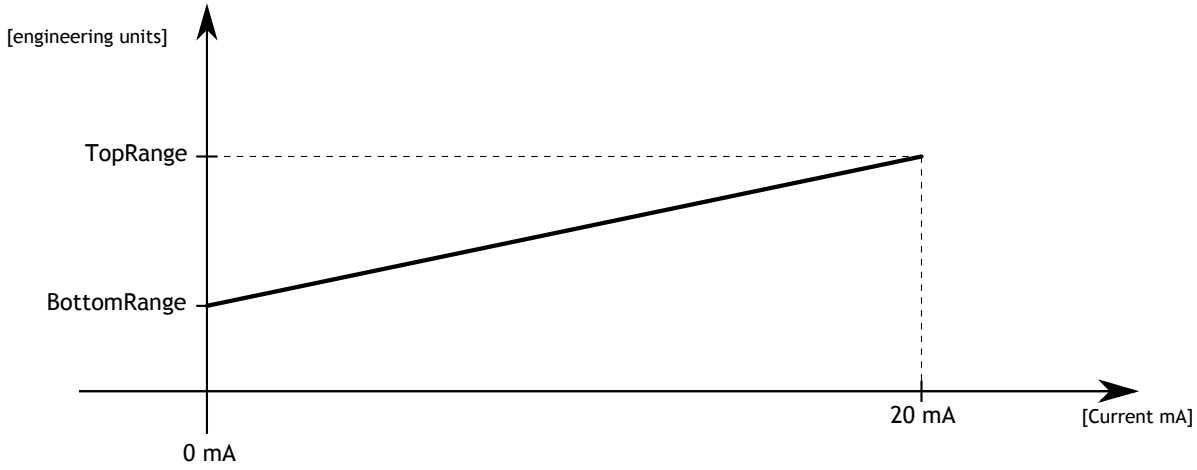
If a sensor reads the presence of CO2 in a space from 0 ppm to 2000 ppm and the electrical signal is 4..20 mA, the parameters are:
"BottomRange" = 0 and "TopRange" = 2000.

- How to do to use library function - :

- With your hands set the related jumper of the input on CURRENT mode. Please see the PIO's datasheet.
- The PIO interface has two universal inputs, enumerated from one to two (UI1, UI2). Passing id = 1 to the function means you are configuring the input number one. Passing id = 2 to the function means you are configuring the input number two.
- Set to one (1) the number of the type of input, using `PIO_UI_Type_Write` function.
- Set the "BottomRange" value, in engineering units, using `PIO_UI_BottomRange_Write` function.
- Set the "TopRange" value, in engineering units, using `PIO_UI_TopRange_Write` function.
- Read the value, in engineering units, from sensor connected to the input using `PIO_UI_Value_Read` function.

Current 0..20 mA universal input

Consider the next figure.



If you must configure an universal input that receives from sensor a signal from 0 mA to 20 mA and the sensor must read temperature from 0 °C to 50 °C, the parameters are:
"BottomRange" = 0 and "TopRange" = 50.

If you must configure an universal input that receives from servomotor a feedback position with 0..20 mA signal and the universal input must read percentage of position from 0% to 100%, the parameters are:
"BottomRange" = 0 and "TopRange" = 100.

If a sensor reads the presence of CO2 in a space, from 0 ppm to 2000 ppm, and the electrical signal is 0..20 mA, the parameters are:
"BottomRange" = 0 and "TopRange" = 2000.

- How to do to use library function - :

- With your hands set the related jumper of the input on CURRENT mode. Please see the PIO's datasheet.
- The PIO interface has two universal inputs, enumerated from one to two (UI1, UI2). Passing id = 1 to the function means you are configuring the input number one. Passing id = 2 to the function means you are configuring the input number two.
- Set to two (2) the number of the type of input, using `PIO_UI_Type_Write` function.
- Set the "BottomRange" value, in engineering units, using `PIO_UI_BottomRange_Write` function.
- Set the "TopRange" value, in engineering units, using `PIO_UI_TopRange_Write` function.
- Read the value, in engineering units, from sensor connected to the input using `PIO_UI_Value_Read` function.

NTC 10K universal input

NTC (negative thermistor coefficient) is a cheap and widespread sensor used for many applications. The sensor has a Ohm output resistance whose value depend from temperature. The sensor gives an output of 10kΩ at 25 °C.

- How to do to use library function - :

- With your hands set the related jumper of the input on NTC mode. Please see the PIO's datasheet.
- The "Bottom" and "Top" parameters must not be used.
- The PIO interface has two universal inputs, enumerated from one to two (UI1, UI2). Passing id = 1 to the function means you are configuring the input number one. Passing id = 2 to the function means you are configuring the input number two.
- Set to three (3) the number of the type of input, using `PIO_UI_Type_Write` function.
- Use `PIO_UI_Value_Read` function to read the value from sensor connected to the input, in engineering units.

DIGITAL universal input

You can use universal input to configure it as digital input.

- How to do to use library function - :

- With your hands set the related jumper of the input on NTC mode. Please see the PIO's datasheet.
- The "Bottom" and "Top" parameters must not be used.
- The PIO interface has two universal inputs, enumerated from one to two (UI1, UI2). Passing id = 1 to the function means you are configuring the input number one. Passing id = 2 to the function means you are configuring the input number two.
- Set to four (4) the number of the type of input, using `PIO_UI_Type_Write` function.
- Use `PIO_UI_Value_Read` function to read the value from the switch connected to the input, 0 for OFF and 1 for ON.

Pio_UI_Type_Write

Declaration `int Pio_UI_Type_Write (int id, int value)`

Description Configures the electrical type for the input channel. See 'UI : General notes'.

Parameter	Type/Description
id	integer number - the number of input channel that must be configured;
value	integer number - the type chosen for the input channel (VOLTAGE, CURRENT 4..20 mA, CURRENT 0..20 mA, NTC or DIGITAL).

Return value integer number - success or error code.

Related functions **Pio_UI_Type_Read**

Platform RASPBIAN on RASPBERRY PI

Return value

+1 for SUCCESS : the write operation is successful.
-1 for error : the write operation is not successful.

Example codes

Ex.1 : Configure Universal Input UI1 to detect signal from Voltage Sensor.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;

    UI_number = 1;
    UI_type = 0;

    Pio_UI_Type_Write (UI_number, UI_type);

    while (1)
    {
        // YOUR CODE
    }
}
```

Ex.2 : Configure Universal Input UI2 to detect signal from 4-20 mA Current Sensor.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;

    UI_number = 2;
    UI_type = 1;

    Pio_UI_Type_Write (UI_number, UI_type);
```

```
    while (1)
    {

        // YOUR CODE
    }
}
```

Ex.3 : Configure Universal Input UI2 to detect signal from Digital Sensor.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;

    UI_number = 2;
    UI_type = 4;

    Pio_UI_Type_Write (UI_number, UI_type);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_UI_Type_Read

Declaration `int Pio_UI_Type_Read (int id)`

Description Reads the type currently configured for the input channel. See 'UI : General notes'.

Parameter Type/Description
id integer number - the number of input channel for which read the type;

Return value integer number - the type of electrical signal currently configured for the input channel.

- 0 : VOLTAGE (factory default)
- 1 : CURRENT 4-20 mA
- 2 : CURRENT 0-20 mA
- 3 : THERMISTOR NTC 10K
- 4 : DIGITAL

Related functions **`Pio_UI_Type_Write`**

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful.
-1 for error : the read operation is not successful.

Example codes

Ex.1 : Read the type of the Universal Input UI1, to know how it is configured.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;

    UI_number = 1;

    UI_type  = Pio_UI_Type_Read (UI_number);
    printf ("The type of input is : %i\n", UI_type);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_UI_BottomVolt_Write

Declaration `int Pio_UI_BottomVolt_Write (int id, int value)`

Description Only when the type for the universal input is "VOLTAGE", set the "BottomVolt" parameter. See 'UI : General notes'.

Parameter Type/Description

id	integer number - the number of input channel that must be configured for "BottomVolt" value;
value	integer number - the value in Volt to set "BottomVolt" parameter.

Return value integer number - success or error code.

Related functions **Pio_UI_BottomVolt_Read**

Platform RASPBIAN on RASPBERRY PI

Parameter: "value"

The value of "BottomVolt" must be inside the range 0..10 Vcc and must be not greater than "TopVolt".

Return value

+1 for SUCCESS : the write operation is successful.
0 for warning : the value of "BottomVolt" is greater than "TopVolt"; however the write operation is successful.
-1 for error : the write operation is not successful.

Example codes

Ex.1 : Configure Universal input UI1 to be VOLTAGE sensor that produces the measure as low as 2 V.

```
int main (void)
{
    Pio_SYSTEM_Init();
```

```
    int UI_number;
    int UI_type;
    int UI_BottomVolt;

    UI_number = 1;
    UI_type = 0;
    UI_BottomVolt = 2;

    Pio_UI_Type_Write (UI_number, UI_type);
    Pio_UI_BottomVolt_Write(UI_number, UI_BottomVolt);

    while (1)
    {
        // YOUR CODE
    }
}
```

Ex.2 : Configure Universal input UI2 to be VOLTAGE sensor that produces the measure as low as 0 V.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;
    int UI_BottomVolt;

    UI_number = 2;
    UI_type = 0;
    UI_BottomVolt = 0;

    Pio_UI_Type_Write (UI_number, UI_type);
    Pio_UI_BottomVolt_Write(UI_number, UI_BottomVolt);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_UI_BottomVolt_Read

Declaration `int Pio_UI_BottomVolt_Read (int id)`

Description Only when the type for the universal input is “VOLTAGE”, reads the “BottomVolt” parameter. See ‘UI : General notes’.

Parameter	Type/Description
id	integer number - the number of input channel for which read the type;

Return value integer number - the value of the “BottomVolt” parameter for the input channel.

Related functions **Pio_UI_BottomVolt_Write**

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful.
-1 for error : the read operation is not successful.

Example codes

Ex.1 : Read the “Bottom” value for the Universal Input UI1.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_BottomVolt;

    UI_number = 1;
    UI_BottomVolt  = Pio_UI_BottomVolt_Read(UI_number);
    printf (“The Bottom voltage value of input is : %i\n”, UI_BottomVolt);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_UI_TopVolt_Write

Declaration `int Pio_UI_TopVolt_Write (int id, int value)`

Description Only when the type for the universal input is “VOLTAGE”, set the “TopVolt” parameter. See ‘UI : General notes’.

Parameter Type/Description
- id — integer number - the number of input channel that must be configured for “TopVolt” value;
- value — integer number - the value in Volt to set “TopVolt” parameter.

Return value integer number - success or error code.

Related functions **Pio_UI_TopVolt_Read**

Platform RASPBIAN on RASPBERRY PI

Parameter: “value”

The value of “TopVolt” must be inside the range 0..10 Vcc and must be greater than “BottomVolt”.

Return value

+1 for SUCCESS : the write operation is successful.
0 for warning : the value of “TopVolt” is less than “BottomVolt”; however the write operation is successful.
-1 for error : the write operation is not successful.

Example codes

Ex.1 : Configure Universal input UI1 to be VOLTAGE sensor that produces the measure between 2 V and 10 V.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;
    int UI_BottomVolt;
    int UI_TopVolt;

    UI_number = 1;
    UI_type = 0;
    UI_BottomVolt = 2;
    UI_TopVolt = 10;

    Pio_UI_Type_Write (UI_number, UI_type);
    Pio_UI_BottomVolt_Write(UI_number, UI_BottomVolt);
    Pio_UI_TopVolt_Write(UI_number, UI_TopVolt);

    while (1)
    {
        // YOUR CODE
    }
}
```

Ex.2 : Configure Universal input UI2 to be VOLTAGE sensor that produces the measure between 4 V and 7 V.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;
    int UI_BottomVolt;
    int UI_TopVolt;

    UI_number = 2;
    UI_type = 0;
    UI_BottomVolt = 4;
    UI_TopVolt = 7;

    Pio_UI_Type_Write (UI_number, UI_type);
    Pio_UI_BottomVolt_Write(UI_number, UI_BottomVolt);
    Pio_UI_TopVolt_Write(UI_number, UI_TopVolt);

    while (1)
    {
        // YOUR CODE
    }
}
```

Ex.3 : Configure Universal input UI1 to be VOLTAGE sensor that produces the measure between 0 V and 10 V.

```
int main (void)
{
    Pio_SYSTEM_Init();
```

```
    int UI_number;
    int UI_type;
    int UI_BottomVolt;
    int UI_TopVolt;

    UI_number = 1;
    UI_type = 0;
    UI_BottomVolt = 0;
    UI_TopVolt = 10;

    Pio_UI_Type_Write (UI_number, UI_type);
    Pio_UI_BottomVolt_Write(UI_number, UI_BottomVolt);
    Pio_UI_TopVolt_Write(UI_number, UI_TopVolt);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_UI_TopVolt_Read

Declaration `int Pio_UI_TopVolt_Read (int id)`

Description Only when the type for the universal input is “VOLTAGE”, read the “TopVolt” parameter. See ‘UI : General notes’.

Parameter Type/Description
id integer number - the number of input channel for which read the type;

Return value integer number - the value of the “TopVolt” parameter for the input channel.

Related functions **Pio_UI_TopVolt_Write**

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful.
-1 for error : the read operation is not successful.

Example codes

Ex.1 : Read the “TopVolt” value for the Universal Input UI1.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_TopVolt;

    UI_number = 1;
    UI_TopVolt  = Pio_UI_TopVolt_Read(UI_number);
    printf (“The Top voltage value of input is : %i\n”, UI_TopVolt);
```

```
    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_UI_BottomRange_Write

Declaration `int Pio_UI_BottomRange_Write (int id, float value)`

Description Only when the type for the universal input is “VOLTAGE” or “CURRENT”, sets the “BottomRange” parameter. See ‘UI : General notes’.

Parameter Type/Description

id	integer number - the number of input channel that must be configured for “BottomRange” value;
value	float number - the value, **in engineering units**, to set “BottomRange” parameter. To get a good measurement precision is recommended not to exceed the range ± 10000 in engineering units. However, any value can be written on the parameter "BottomRange."

Return value integer number - success or error code.

Related functions **Pio_UI_BottomRange_Read**

Platform RASPBIAN on RASPBERRY PI

Return value

+1 for SUCCESS : the write operation is successful.
-1 for error : the write operation is not successful.

Examples codes

Ex.1 : Configure Universal input UI1 to be VOLTAGE sensor that produces the measure between 0 and 10 V to read, in engineering units, a temperature from 0 to 50 °C.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;
    int UI_BottomVolt;
    int UI_TopVolt;
    float UI_BottomRange;

    UI_number = 1;
    UI_type = 0;
    UI_BottomVolt = 2;
    UI_TopVolt = 10;
    UI_BottomRange = 0;

    Pio_UI_Type_Write (UI_number, UI_type);
    Pio_UI_BottomVolt_Write(UI_number, UI_BottomVolt);
```

```
    Pio_UI_TopVolt_Write(UI_number, UI_TopVolt);
    Pio_UI_BottomRange_Write(UI_number, UI_BottomRange);

    while (1)
    {
        // YOUR CODE
    }
}
```

Ex.2 : Configure Universal input UI2 to be CURRENT 4..20 mA sensor, to read, in engineering units, a pressure from 20 to 500 Pa.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;
    float UI_BottomRange;

    UI_number = 2;
    UI_type = 1;
    UI_BottomRange = 20;

    Pio_UI_Type_Write (UI_number, UI_type);
    Pio_UI_BottomRange_Write(UI_number, UI_BottomRange);

    while (1)
    {
        // YOUR CODE
    }
}
```

Ex.3 : Configure Universal input UI2 to be CURRENT 0..20 mA sensor, to read, in engineering units, humidity from 0 to 100 %.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;
    float UI_BottomRange;

    UI_number = 2;
    UI_type = 2;
    UI_BottomRange = 0;

    Pio_UI_Type_Write (UI_number, UI_type);
    Pio_UI_BottomRange_Write(UI_number, UI_BottomRange);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_UI_BottomRange_Read

Declaration `float Pio_UI_BottomRange_Read (int id)`

Description Only when the type for the universal input is “VOLTAGE” or “CURRENT”, reads the “BottomRange” parameter. See ‘UI : General notes’.

Parameter Type/Description
id integer number - the number of input channel for which read the “BottomRange”;

Return value float number - the value of the “BottomRange” parameter, **in engineering units**, for the input channel.

Related functions **Pio_UI_TopRange_Write**

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful.
-1111111 for error : the read operation is not successful.

Example codes

Ex.1 : Read the “BottomRange” value for Universal Input UI1.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    float UI_BottomRange;

    UI_number = 1;
    UI_BottomRange  = Pio_UI_BottomRange_Read(UI_number);
    printf (“The BottomRange value of input is : %f\n”, UI_BottomRange);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_UI_TopRange_Write

Declaration `int Pio_UI_TopRange_Write (int id, float value)`

Description Only when the type for the universal input is “VOLTAGE” or “CURRENT”, sets the “TopRange” parameter. See ‘UI : General notes’.

Parameter Type/Description
id integer number - the number of input channel that must be configured for “TopRange” value;
value float number - the value, **in engineering units**, to set “TopRange” parameter. To get a good measurement precision is recommended not to exceed the range ±

10000 in engineering units. However, any value can be written on the parameter "TopRange."

Return value integer number - success or error code.

Related functions **Pio_UI_TopRange_Read**

Platform RASPBIAN on RASPBERRY PI

Return value

+1 for SUCCESS : the write operation is successful.
-1 for error : the write operation is not successful.

Examples codes

Ex.1 : Configure Universal input UI1 to be VOLTAGE that produces the measure between 2 and 10 V to read, in engineering units, a temperature from 0 to 50 °C.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;
    int UI_BottomVolt;
    int UI_TopVolt;
    float UI_BottomRange;
    float UI_TopRange;

    UI_number = 1;
    UI_type = 0;
    UI_BottomVolt = 2;
    UI_TopVolt = 10;
    UI_BottomRange = 0;
    UI_TopRange = 50;

    Pio_UI_Type_Write (UI_number, UI_type);
    Pio_UI_BottomVolt_Write(UI_number, UI_BottomVolt);
    Pio_UI_TopVolt_Write(UI_number, UI_TopVolt);
    Pio_UI_BottomRange_Write(UI_number, UI_BottomRange);
    Pio_UI_TopRange_Write(UI_number, UI_TopRange);

    while (1)
    {
        // YOUR CODE
    }
}
```

Ex.2 : Configure Universal input UI2 to be CURRENT 4..20 mA sensor, to read, in engineering units, a pressure from 20 to 500 Pa.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;
    float UI_BottomRange;
```

```
    float UI_TopRange;

    UI_number = 2;
    UI_type = 1;
    UI_BottomRange = 20;
    UI_TopRange = 500;

    Pio_UI_Type_Write (UI_number, UI_type);
    Pio_UI_BottomRange_Write(UI_number, UI_BottomRange);
    Pio_UI_Toprange_Write(UI_number, UI_TopRange);

    while (1)
    {
        // YOUR CODE
    }
}
```

Ex.3 : Configure Universal input UI2 to be CURRENT 0..20 mA sensor, to read, in engineering units, humidity from 0 to 100 %.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;
    float UI_BottomRange;
    float UI_TopRange;

    UI_number = 2;
    UI_type = 2;
    UI_BottomRange = 0;
    UI_TopRange = 100;

    Pio_UI_Type_Write (UI_number, UI_type);
    Pio_UI_BottomRange_Write(UI_number, UI_BottomRange);
    Pio_UI_TopRange_Write(UI_number, UI_TopRange);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_UI_TopRange_Read

Declaration `float Pio_UI_TopRange_Read (int id)`

Description Only when the type for the universal input is “VOLTAGE” or “CURRENT”, reads the “TopRange” parameter. See ‘UI : General notes’.

Parameter Type/Description
id integer number - the number of input channel for which read the “TopRange”;

Return value float number - the value of the “TopRange” parameter, **in engineering units**, for the input channel.

Related functions **Pio_UI_TopRange_Write**

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful.
-1111111 for error : the read operation is not successful.

Example codes

Ex.1 : Read the “TopRange” value for Universal Input UI2.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    float UI_TopRange;

    UI_number = 2;
    UI_TopRange  = Pio_UI_TopRange_Read(UI_number);

    printf (“The TopRange value of input is : %f\n”, UI_TopRange);
    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_UI_Value_Read

Declaration `float Pio_UI_Value_Read (int id)`

Description Reads the value from the universal input channel.

Parameter Type/Description
id integer number - the number of input channel for which read the value.

Return value float number - the value, **in engineering units**, for the universal input channel.

Related functions None.

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful.
-1111111 for error: the read operation is not successful.
-2222222 for sensor not physically connected to the PIO’s terminal if the input is configured as 4..20 mA.
-3333333 for sensor not physically connected to the PIO’s terminal or for measurement out of range if the input is configured as NTC 10K.

Example codes

Ex.1 : Configure Universal input UI1 to be VOLTAGE that produces the measure between 2 V and 10 V to read, in engineering units, a temperature from 0 to 50 °C. (Please see UI : General notes for hardware settings).

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;
    int UI_BottomVolt;
    int UI_TopVolt;
    float UI_BottomRange;
    float UI_TopRange;
    float UI_measure;

    UI_number = 1;
    UI_type = 0;
    UI_BottomVolt = 2;
    UI_TopVolt = 10;
    UI_BottomRange = 0;
    UI_TopRange = 50;

    Pio_UI_Type_Write (UI_number, UI_type);
    Pio_UI_BottomVolt_Write(UI_number, UI_BottomVolt);
    Pio_UI_TopVolt_Write(UI_number, UI_TopVolt);
    Pio_UI_BottomRange_Write(UI_number, UI_BottomRange);
    Pio_UI_TopRange_Write(UI_number, UI_TopRange);

    while (1)
    {
        UI_measure = Pio_UI_Value_Read(UI_number);
        printf ("The measure for input is : %f\n", UI_measure);
        // YOUR CODE
    }
}
```

Ex.2 : Configure Universal input UI2 to be CURRENT 4..20 mA sensor, to read, in engineering units, a pressure from 20 to 500 Pa. (Please see UI : General notes for hardware settings).

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;
    float UI_BottomRange;
    float UI_TopRange;
    float UI_measure;

    UI_number = 2;
    UI_type = 1;
    UI_BottomRange = 20;
    UI_TopRange = 500;

    Pio_UI_Type_Write (UI_number, UI_type);
    Pio_UI_BottomRange_Write(UI_number, UI_BottomRange);
    Pio_UI_TopRange_Write(UI_number, UI_TopRange);

    while (1)
    {
```

```
            UI_measure = Pio_UI_Value_Read(UI_number);
            printf ("The measure for input is : %f\n", UI_measure);
            // YOUR CODE
        }
    }
```

Ex.3 : Configure Universal input UI2 to be CURRENT 0..20 mA sensor, to read, in engineering units, humidity from 0 to 100 %. (Please see UI : General notes for hardware settings).

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;
    float UI_BottomRange;
    float UI_TopRange;
    float UI_measure;

    UI_number = 2;
    UI_type = 2;
    UI_BottomRange = 0;
    UI_TopRange = 100;

    Pio_UI_Type_Write (UI_number, UI_type);
    Pio_UI_BottomRange_Write(UI_number, UI_BottomRange);
    Pio_UI_TopRange_Write(UI_number, UI_TopRange);

    while (1)
    {
        UI_measure = Pio_UI_Value_Read(UI_number);
        printf ("The measure for input is : %f\n", UI_measure);
        // YOUR CODE
    }
}
```

Ex.4 : Configure Universal input UI1 to be NTC 10K sensor, to read temperature from -30 to +110 °C. (Please see UI : General notes for hardware settings).

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;
    float UI_measure;

    UI_number = 1;
    UI_type = 3;

    Pio_UI_Type_Write (UI_number, UI_type);

    while (1)
    {
        UI_measure = Pio_UI_Value_Read(UI_number);
        printf ("The measure for input is %f\n", UI_measure);
        // YOUR CODE
    }
}
```

Ex.5 : Configure Universal input UI2 to be DIGITAL sensor, to read status OFF or ON. (Please see UI : General notes for hardware settings).

```
int main (void)
{
    Pio_SYSTEM_Init();

    int UI_number;
    int UI_type;
    int UI_status;

    UI_number = 2;
    UI_type = 4;

    Pio_UI_Type_Write (UI_number, UI_type);

    while (1)
    {
        UI_status = (int) Pio_UI_Value_Read(UI_number);
        printf (“The status for input is : %i\n”, UI_status);
        // YOUR CODE
    }
  }
```

DIGITAL INPUT LIBRARY : PIO_DI

- `int Pio_DI_Status_Read (int id)`

DI : General notes

The library of the digital Inputs (DI) is a set of software functions you use to read the status of a switch connected to the input, that may be open (OFF = 0) or closed (ON = 1).
In the PIO interface there is a service button that you can use for generic purposes. It is an additional digital input named DI5. On the PIO's data-sheet the button is named SW1.

- How to do to use library function - :

- No hardware settings to do with your hands on the PIO interface;
- The PIO interface has four opto-isolated digital inputs, enumerated from one to four (DI1, DI2, DI3, DI4). Passing id = 1 to the function means you are configuring the input number one. Passing id = 2 to the function means you are configuring the input number two and so on.
- Use `PIO_DI_Status_Read` function to read the status of the switch connected to the input.

Pio_DI_Status_Read

Declaration `int Pio_DI_Status_Read (int id)`

Description Reads the status from the digital input channel.

Parameter Type/Description
id integer number - the number of input channel for which read the status.

Return value integer number - the status of digital input, 0 for OFF (open contact) and 1 for ON (close contact).

Related functions None.

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful.
-1 for error : the read operation is not successful.

Example codes

Ex.1 : Read the status of DI3 digital input. (Please see PIO datasheet for electrical connections).

```
int main (void)
{
    Pio_SYSTEM_Init();

    int DI_number;
    int DI_status;

    DI_number = 3;

    while (1)
    {
        DI_status = Pio_DI_Status_Read(DI_number);
        printf ("The status for input is : %i\n", DI_status);
        // YOUR CODE
    }
}
```

Ex.2 : Read the status of DI4 digital input. (Please see PIO datasheet for electrical connections).

```
int main (void)
{
    Pio_SYSTEM_Init();

    int DI_number;
    int DI_status;

    DI_number = 4;

    while (1)
    {
        DI_status = Pio_DI_Status_Read(DI_number);
        printf ("The status for input is : %i\n", DI_status);
        // YOUR CODE
```

```
    }
}
```

Ex.3 : Read the status of DI1 digital input. (Please see PIO datasheet for electrical connections).

```
int main (void)
{
    Pio_SYSTEM_Init();

    int DI_number;
    int DI_status;

    DI_number = 1;

    while (1)
    {
        DI_status = Pio_DI_Status_Read(DI_number);
        printf (“The status for input is : %i\n”, DI_status);
        // YOUR CODE
    }
}
```

Ex.4 : Verifies if the service button SW1 is pressed.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int DI_number;
    int DI_status;

    DI_number = 5;

    while (1)
    {
        DI_status = Pio_DI_Status_Read(DI_number);
        printf (“The status for input is : %i\n”, DI_status);
        // YOUR CODE
    }
}
```

ANALOGUE OUTPUT LIBRARY : PIO_AO

- `int Pio_AO_BottomVolt_Write(int id, int value);`
- `int Pio_AO_BottomVolt_Read(int id);`
- `int Pio_AO_TopVolt_Write(int id, int value);`
- `int Pio_AO_TopVolt_Read(int id);`
- `int Pio_AO_Value_Write (int id, int value);`
- `int Pio_AO_Value_Read(int id);`

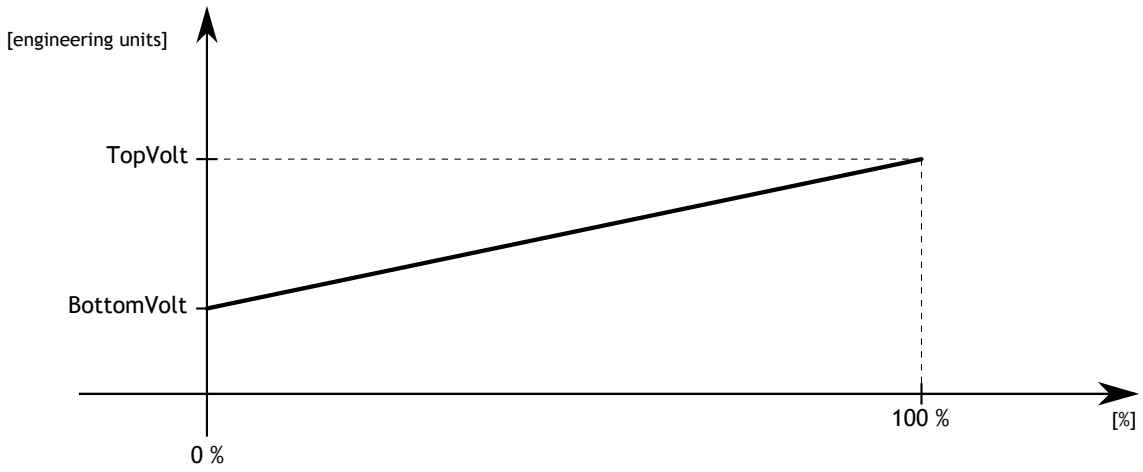
AO : General notes

The library of the Analogue Outputs (AO) is a set of software functions that you must use to control the analogue outputs of the PIO interface.

To understand how to use the PIO_AO library you need to familiarize with 2 parameters that are:
1. "BottomVolt"
2. "TopVolt".

The channels AO1 and AO2 provide in output a voltage signal, detectable with measuring instrument for voltage, configurable between "BottomVolt" and TopVolt". The value of "BottomVolt" in output corresponds to the 0% percentage whilst the value of "TopVolt" corresponds to the 100%. So, with a number between 0 and 100 %, you can pilot any voltage between "BottomVolt" and "TopVolt".

Consider the next figure.



If you must control an actuator with a command from 2 to 10 Volts the parameters are:
"BottomVolt" = 2, "TopVolt" = 10. With these parameters, when you set 0% using the function `Pio_AO_Value_Write`, the output value will be 2 Volts and when you set 100% the output value will be 10 Volts. If you set 50% the output will be 6 Volts.

If, for example, you must control a damper with a command from 0 to 10 Volts the parameters are:
"BottomVolt" = 0, "TopVolt" = 10. With these parameters, when you set 0% using the function `Pio_AO_Value_Write`, the output value will be 0 Volts and when you set 100% the output value will be 10 Volts. If you set 50% the output will be 5 Volts.

Another example. If you must control a servomotor with a command from 4 to 8 Volts the parameters are: "BottomVolt" = 4, "TopVolt" = 8. With these parameters, when you set 0% using the function `Pio_AO_Value_Write`, the output value will be 4 Volts and when you set 100% the output value will be 8 Volts. If you set 50% the output will be 6 Volts.

- How to do to use library function - :

- No hardware settings to do with your hands on the PIO interface;
- The PIO interface has two analogue outputs, enumerated from one to two (AO1, AO2). Passing id = 1 to the function means you are configuring the output number one. Passing id = 2 to the function means you are configuring the output number two.
- Set the "BottomVolt" value using `PIO_AO_BottomVolt_Write` function.
- Set the "TopVolt" value using `PIO_AO_TopVolt_Write` function.
- Be sure "TopVolt" is greater than "BottomVolt".
- You write the value you want, **in the range 0%..100%**, to control the output using `Pio_AO_Value_Write` function.
- You can read the settings in use by using `Pio_AO_BottomVolt_Read`, `Pio_AO_TopVolt_Read` and `Pio_AO_Value_Read`.

Pio_AO_BottomVolt_Write

Declaration `int Pio_AO_BottomVolt_Write (int id, int value)`

Description Sets the “BottomVolt” parameter. See ‘AO : General notes’.

Parameter Type/Description
- id — integer number - the number of output channel that must be configured for “BottomVolt” value;
- value — integer number - the value in Volt to set “BottomVolt” parameter.

Return value integer number - success or error code.

Related functions **`Pio_AO_BottomVolt_Read`**

Platform RASPBIAN on RASPBERRY PI

Parameter: “value”

The value of “BottomVolt” must be inside the range 0..10 Vcc and must be not greater than “TopVolt”.

Return value

+1 for SUCCESS : the write operation is successful.
0 for warning : the value of “BottomVolt” is greater than “TopVolt”; however the write operation is successful.
-1 for error : the write operation is not successful.

Example codes

Ex.1 : Configure AO1 to control an actuator that receives the command from 2 V to 10 V.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int AO_number;
    int AO_BottomVolt;

    AO_number = 1;
    AO_BottomVolt = 2;

    Pio_AO_BottomVolt_Write(AO_number, AO_BottomVolt);

    while (1)
    {
        // YOUR CODE
    }
}
```

Ex.2 : Configure AO2 to control a damper from 0 V to 10 V.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int AO_number;
```

```
    int AO_BottomVolt;

    AO_number = 2;
    AO_BottomVolt = 0;

    Pio_AO_BottomVolt_Write(AO_number, AO_BottomVolt);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_AO_BottomVolt_Read

Declaration `int Pio_AO_BottomVolt_Read (int id)`

Description Reads the “BottomVolt” parameter. See ‘AO : General notes’.

Parameter
id — Type/Description: integer number - the number of output channel for which read the “BottomVolt” value.

Return value integer number - the value of the “BottomVolt” parameter for the output channel.

Related functions **Pio_AO_BottomVolt_Write**

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful.
-1 for error : the read operation is not successful.

Example codes

Ex.1 : Read the actual “BottomVolt” value for output AO1.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int AO_number;
    int AO_BottomVolt;

    AO_number = 1;
    AO_BottomVolt  = Pio_AO_BottomVolt_Read(AO_number);

    printf (“The Bottom voltage value of output is : %i\n”, AO_BottomVolt);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_AO_TopVolt_Write

Declaration `int Pio_AO_TopVolt_Write (int id, int value)`

Description Set the “TopVolt” parameter. See ‘AO : General notes’.

Parameter Type/Description
- id — integer number - the number of output channel that must be configured for “TopVolt” value;
- value — integer number - the value in Volt to set “TopVolt” parameter.

Return value integer number - success or error code.

Related functions **`Pio_AO_TopVolt_Read`**

Platform RASPBIAN on RASPBERRY PI

Parameter: “value”

The value of “TopVolt” must be inside the range 0..10 Vcc and must be greater than “BottomVolt”.

Return value

+1 for SUCCESS : the write operation is successful.
0 for warning : the value of “TopVolt” is not greater than “BottomVolt”; however the write operation is successful.
-1 for error : the write operation is not successful.

Example codes

Ex.1 : Configure AO1 to control an actuator that receives the command from 2 V to 10 V.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int AO_number;
    int AO_BottomVolt;
    int AO_TopVolt;

    AO_number = 1;
    AO_BottomVolt = 2;
    AO_TopVolt = 10;

    Pio_AO_BottomVolt_Write(AO_number, AO_BottomVolt);
    Pio_AO_TopVolt_Write(AO_number, AO_TopVolt);

    while (1)
    {
        // YOUR CODE
    }
}
```

Ex.2 : Configure AO2 to control a damper from 0 V to 10 V.

```
int main (void)
{
```

```
    Pio_SYSTEM_Init();

    int AO_number;
    int AO_BottomVolt;
    int AO_TopVolt;

    AO_number = 2;
    AO_BottomVolt = 0;
    AO_TopVolt = 10;

    Pio_AO_BottomVolt_Write(AO_number, AO_BottomVolt);
    Pio_AO_TopVolt_Write(AO_number, AO_TopVolt);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_AO_TopVolt_Read

Declaration `int Pio_AO_TopVolt_Read (int id)`

Description Read the “TopVolt” parameter. See ‘AO : General notes’.

Parameter

	Type/Description
id	integer number - the number of output channel for which read the “TopVolt” value.

Return value integer number - the value of the “TopVolt” parameter for the output channel.

Related functions **Pio_AO_TopVolt_Write**

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful.
-1 for error : the read operation is not successful.

Example codes

Ex.1 : Read the actual “TopVolt” value for output AO2.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int AO_number;
    int AO_TopVolt;

    AO_number = 2;
    AO_TopVolt  = Pio_AO_TopVolt_Read(AO_number);

    printf (“The Top voltage value of output is %i\n”, AO_TopVolt);

    while (1)
```

```
    {
        // YOUR CODE
    }
}
```

Pio_AO_Value_Write

Declaration `int Pio_AO_Value_Write (int id, int value)`

Description Sets the value to control from 0% to 100%, on the output channel, inside the range from “BottomVolt” to “TopVolt”. See ‘AO : General notes’.

Parameter	Type/Description
id	integer number - the number of output channel that must be configured.
value	integer number - the value to control the output.

Return value integer number - success or error code.

Related functions **Pio_AO_Value_Read**

Platform RASPBIAN on RASPBERRY PI

Parameter: “value”

“value” must be inside the range **0..100%**.

Return value

+1 for SUCCESS : the write operation is successful.
-1 for error : the write operation is not successful.

Example codes

Ex.1 : Configure AO1 to control an actuator from 2 V to 10 V. Set the command value to have an output of 9 V.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int AO_number;
    int AO_BottomVolt;
    int AO_TopVolt;
    int AO_command;

    AO_number = 1;
    AO_BottomVolt = 2;
    AO_TopVolt = 10;
    AO_command = 87;

    Pio_AO_BottomVolt_Write(AO_number, AO_BottomVolt);
    Pio_AO_TopVolt_Write(AO_number, AO_TopVolt);

    while (1)
    {
        Pio_AO_Value_Write(AO_number, AO_command);
```

```
        // YOUR CODE
    }
}
```

Ex.2 : Configure AO2 to control a damper from 0 V to 10 V. Set the command value to have an output of 7 V.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int AO_number;
    int AO_BottomVolt;
    int AO_TopVolt;
    int AO_command;

    AO_number = 2;
    AO_BottomVolt = 0;
    AO_TopVolt = 10;
    AO_command = 70;

    Pio_AO_BottomVolt_Write(AO_number, AO_BottomVolt);
    Pio_AO_TopVolt_Write(AO_number, AO_TopVolt);

    while (1)
    {
        Pio_AO_Value_Write(AO_number, AO_command);
        // YOUR CODE
    }
}
```

Pio_AO_Value_Read

Declaration `int Pio_AO_Value_Read (int id)`

Description Reads the value of the actual control position for the output channel. See 'AO : General notes'.

Parameter Type/Description
id integer number - the number of output channel for which read the value.

Return value integer number - the actual value of the output channel.

Related functions **`Pio_AO_Value_Write`**

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful.
-1 for error : the read operation is not successful.

Example codes

Ex.1 : Read the actual position of output AO2.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int AO_number;
    int AO_actual_position;

    AO_number = 2;
    AO_actual_position = Pio_AO_Value_Read(AO_number);
    printf (“The percent position of output is : %i\n”, AO_actual_position);

    while (1)
    {
        // YOUR CODE
    }
}
```

DIGITAL OUTPUT LIBRARY : PIO_DO

- `int Pio_DO_Command_Write(int id, int value);`
- `int Pio_DO_Command_Read(int id);`

DO : General notes

The library of the digital Outputs (DO) is a set of software functions you use to write command ON (1) or OFF (0) to the digital output to energise/de-energise the corresponding relay. So you can control pumps, the START/STOP command for inverters, fans etc.

- How to do to use library function - :

- No hardware settings to do with your hands on the PIO interface;
- The PIO interface has two digital outputs with relays, enumerated from one to two (DO1, DO2). Passing id = 1 to the function means you are configuring the output number one. Passing id = 2 to the function means you are configuring the output number two.
- To turn ON a digital output you must write the integer value “1” using the function `Pio_DO_Command_Write`.
- To turn OFF a digital output you must write the integer value “0” using the function `Pio_DO_Command_Write`.

Pio_DO_Command_Write

Declaration `int Pio_DO_Command_Write (int id, int value)`

Description Sets the value to 0 (zero) or 1 (one) to control OFF or ON the output channel. See 'DO : General notes'.

Parameter Type/Description

id	integer number - the number of output channel that must be configured to be OFF or ON;
value	integer number - the value to control the output (0=OFF=OPEN contact), (1=ON=CLOSE contact).

Return value integer number - success or error code.

Related functions **none**

Platform RASPBIAN on RASPBERRY PI

Return value

+1 for SUCCESS : the write operation is successful.
-1 for error : the write operation is not successful.

Example codes

Ex.1 : Configure DO1 to turn ON a pump.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int DO_number;
    int DO_command_ON;

    DO_number = 1;
    DO_command_ON = 1;

    while (1)
    {
        Pio_DO_Command_Write(DO_number, DO_command_ON);
        // YOUR CODE
    }
}
```

Ex.2 : Configure DO2 to START a fan.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int DO_number;
    int DO_command_ON;

    DO_number = 2;
    DO_command_ON = 1;

    while (1)
```

```
    {
        Pio_DO_Command_Write(DO_number, DO_command_ON);
        // YOUR CODE
    }
}
```

Ex.3 : Configure DO2 to turn OFF a light.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int DO_number;
    int DO_command_OFF;

    DO_number = 2;
    DO_command_OFF = 0;

    while (1)
    {
        Pio_DO_Command_Write(DO_number, DO_command_OFF);
        // YOUR CODE
    }
}
```

Pio_DO_Command_Read

Declaration `int Pio_DO_Command_Read (int id)`

Description Reads the value from digital output current state. See 'DO : General notes'.

Parameter Type/Description
id integer number - the number of output channel that must be configured to be OFF or ON;

Return value integer number - success or error code.

Related functions **none**

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful. Value can be 0=OFF=OPEN or 1=ON=CLOSE.
-1 for error : the read operation is not successful.

Ex.1 : Read the actual status for Relay 1.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int DO_number;
    int DO_status;

    DO_number = 1;
```

```
    DO_status = Pio_DO_Command_Read(1);
    printf ("The status of DO1 output is : %i\n", DO_status);

    while (1)
    {
        // YOUR CODE
    }
}
```

LED LIBRARY : PIO_LED

- `int Pio_LED_Command_Write(int id, int value);`

LED General notes

The library of Led (LED) is a set of software functions you use to turn ON or OFF one of three Leds. PIO interface has 3 Leds, one red and two green. They can be used to give some visual information about your application.

- How to do to use library function - :

- No hardware settings to do with your hands on the PIO interface;
- The PIO interface has three leds, enumerated from one to three (LED1, LED2, LED3). Passing id = 1 to the function means you are configuring the first green Led. Passing id = 2 to the function means you are configuring the second green Led. Passing id = 3 to the function means you are configuring the red Led.
- To turn ON a Led you must write the integer value “1” using `Pio_LED_Command_Write`.
- To turn OFF a Led you must write the integer value “0” using `Pio_LED_Command_Write`.

Pio_LED_Command_Write

Declaration `int Pio_LED_Command_Write (int id, int value)`

Description Turn OFF or ON the chosen Led. See 'LED : General notes'.

Parameter Type/Description
id integer number - the number of LED that must be configured to be ON or OFF;
value integer number - the value to control the Led (0=Led OFF), (1=Led ON).

Return value integer number - success or error code.

Related functions **none**

Platform RASPBIAN on RASPBERRY PI

Return value

+1 for SUCCESS : the write operation is successful.
-1 for error : the write operation is not successful.

Example codes

Ex.1 : Configure the red Led to be blinking every 500 ms.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int LED_number;
    int LED_ON;
    int LED_OFF,

    LED_number = 3;
    LED_OFF = 0;
    LED_ON = 1;

    while (1)
    {
        Pio_LED_Command_Write(LED_number, LED_ON);
        delay(500);
        Pio_LED_Command_Write(LED_number, LED_OFF);
        delay(500);
        // YOUR CODE
    }
}
```

Ex.2 : Configure the green Led 1 to be turned ON when digital input DI1 is ON.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int DI_number;
    int DI_status;
    int LED_number;
    int LED_ON;
    int LED_OFF,
```

```
    DI_number = 1;
    LED_number = 1;
    LED_OFF = 0;
    LED_ON = 1;

    while (1)
    {
        DI_status = Pio_DI_Status_Read(DI_number);
        if (DI_status == 0) {Pio_LED_Command_Write(LED_number, LED_OFF);}
            else { Pio_LED_Command_Write(LED_number, LED_ON);}
        // YOUR CODE
    }
}
```

RTC (Real Time Clock) LIBRARY : PIO_RTC

```
int Pio_RTC_Start(void);
int Pio_RTC_Stop(void);
int Pio_RTC_Year_Write(int value);
int Pio_RTC_Year_Read(void);
int Pio_RTC_Month_Write(int value);
int Pio_RTC_Month_Read(void);
int Pio_RTC_DayOfMonth_Write(int value);
int Pio_RTC_DayOfMonth_Read(void);
int Pio_RTC_DayOfWeek_Write(int value);
int Pio_RTC_DayOfWeek_Read(void);
int Pio_RTC_Hour_Write(int value);
int Pio_RTC_Hour_Read(void);
int Pio_RTC_Minute_Write(int value);
int Pio_RTC_Minute_Read(void);
int Pio_RTC_Second_Write(int value);
int Pio_RTC_Second_Read(void);
```

RTC : General notes

The library of the Real Time Clock (RTC) is a set of software functions you use to set an internal clock in the PIO interface. When the clock is running you can read the current date and time for any use you need.

The RTC device on PIO interface has a backup battery installed. From the factory the battery and clock are disabled. The only way to start the clock and enable the backup battery is the use of function `Pio_RTC_Start`. When you turn OFF the power for PIO, the clock will still running anyway because the battery, after calling `Pio_RTC_Start` before the shut down of the interface, was been enabled. Otherwise, after a new power ON, the clock remains stopped. To put clock in the factory conditions (clock and battery disabled) you must use the function `Pio_RTC_Stop`.

- How to do to use library function - :

- No hardware settings to do with your hands on the PIO interface;
- Enable RTC using `Pio_RTC_Start`.

Pio_RTC_Start

Declaration `int Pio_RTC_Start(void)`

Description Starts the clock time counting, from previous settings of date and time. Also enables the battery's backup. See 'RTC : General notes'.

Parameter Type/Description
none

Return value integer number – success, warning or error code.

Related functions **Pio_RTC_Stop**

Platform RASPBIAN on RASPBERRY PI

Return value

+1 for SUCCESS : the start operation is successful.
0 for warning : RTC is just running (the function was called in a previous INIT procedure).
-1 for error : the start operation is not successful.

Example codes

Ex.1 : Enable RTC to start the time counting.

```
int main (void)
{
    int RTC_status;

    Pio_SYSTEM_Init();
    Pio_RTC_Start();

    while (1)
    {
        // YOUR CODE
    }
}
```

Ex.2 : Enable RTC to start time counting.

```
int main (void)
{
    int RTC_status;

    Pio_SYSTEM_Init();
    RTC_status = Pio_RTC_Start();

    If (RTC_status == 1;) { printf ("The clock is started successfully %i\n");}
    If (RTC_status == 0;) { printf ("The clock was just started %i\n");}
    If (RTC_status == -1;) { printf ("The clock is not started %i\n");}

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_RTC_Stop

Declaration `int Pio_RTC_Stop(void)`

Description Stops the clock time counting and puts RTC's configuration at factory state, disabling battery backup. See 'RTC : General notes'.

Parameter Type/Description
none

Return value integer number – success, or error code.

Related functions **Pio_RTC_Start**

Platform RASPBIAN on RASPBERRY PI

Return value

+1 for SUCCESS : the stop operation is successful.
-1 for error : the stop operation is not successful.

Example codes

Ex.1 : Set RTC to factory state (clock not running and battery backup disabled).

```
int main (void)
{
    Pio_SYSTEM_Init();
    Pio_RTC_Stop();

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_RTC_Year_Write

Declaration `int Pio_RTC_Year_Write(int value)`

Description Sets the year on RTC.

Parameter Type/Description
value integer number - the value of current year.

Return value integer number – success or error code.

Related functions **Pio_RTC_Year_Read**

Platform RASPBIAN on RASPBERRY PI

Parameter: "value"

“value” must be inside the range 0..99.

Return value

+1 for SUCCESS : the write operation is successful.
-1 for error : the write operation is not successful.

Example codes

Ex.1 : Write 2014 to RTC current year.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int RTC_year;

    RTC_year = 14;

    Pio_RTC_Year_Write(RTC_year);

    while (1)
    {
        // YOUR CODE
    }
  }
```

Pio_RTC_Year_Read

Declaration `int Pio_RTC_Year_Read(void)`

Description Reads current year from RTC.

Parameter Type/Description
none

Return value integer number – the current year’s value.

Related functions **`Pio_RTC_Year_Write`**

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful.
-1 for error : the read operation is not successful.

Example codes

Ex.1 : Read current year from RTC.

```
int main (void)
{
    Pio_SYSTEM_Init();
```

```
    int RTC_year;

    RTC_year = Pio_RTC_Year_Read();
    printf (“The current year is : %i\n”, RTC_year);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_RTC_Month_Write

Declaration `int Pio_RTC_Month_Write(int value)`

Description Sets the month on RTC.

Parameter Type/Description
value integer number - the current month.

Return value integer number – success or error code.

Related functions **Pio_RTC_Month_Read**

Platform RASPBIAN on RASPBERRY PI

Parameter: “value”

“value” must be inside the range 1..12.

Return value

+1 for SUCCESS : the write operation is successful.
-1 for error : the write operation is not successful.

Example codes

Ex.1 : Write ‘April 2014’ to RTC..

```
int main (void)
{
    Pio_SYSTEM_Init();

    int RTC_year;
    int RTC_month;

    RTC_year = 14;
    RTC_month = 4;

    Pio_RTC_Year_Write(RTC_year);
    Pio_RTC_Month_Write(RTC_month);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_RTC_Month_Read

Declaration `int Pio_RTC_Month_Read(void)`

Description Reads current month's value from RTC.

Parameter Type/Description
none

Return value integer number – the current month.

Related functions **Pio_RTC_Month_Write**

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful.
-1 for error : the read operation is not successful.

Example codes

Ex.1 : Read current month from RTC.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int RTC_month;

    RTC_month = Pio_RTC_Month_Read();
    printf ("The current month is : %i\n", RTC_month);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_RTC_DayOfMonth_Write

Declaration `int Pio_RTC_DayOfMonth_Write(int value)`

Description Sets the day of month on RTC.

Parameter Type/Description
value integer number - the current day of month.

Return value integer number – success or error code.

Related functions **Pio_RTC_DayOfMonth_Read**

Platform RASPBIAN on RASPBERRY PI

Parameter: “value”

“value” must be inside the range 1..31.

Return value

+1 for SUCCESS : the write operation is successful.
-1 for error : the write operation is not successful.

Example codes

Ex.1 : Write ’25 April 2014’ to RTC.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int RTC_year;
    int RTC_month;
    int RTC_dayofmonth;

    RTC_year = 14;
    RTC_month = 4;
    RTC_dayofmonth = 25;

    Pio_RTC_Year_Write(RTC_year);
    Pio_RTC_Month_Write(RTC_month);
    Pio_RTC_DayOfMonth_Write(RTC_dayofmonth);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_RTC_DayOfMonth_Read

Declaration `int Pio_RTC_DayOfMonth_Read(void)`

Description Reads current day of month from RTC.

Parameter Type/Description
none

Return value integer number – the current day of month.

Related functions **`Pio_RTC_DayOfMonth_Write`**

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful.
-1 for error : the read operation is not successful.

Example codes

Ex.1 : Read current day of month from RTC.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int RTC_dayofmonth;

    RTC_dayofmonth = Pio_RTC_DayOfMonth_Read();
    printf ("The current day of month is : %i\n", RTC_dayofmonth);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_RTC_DayOfWeek_Write

Declaration `int Pio_RTC_DayOfWeek_Write(int value)`

Description Sets the day of week on RTC.

Parameter Type/Description
value integer number - the current day of week.

Return value integer number – success or error code.

Related functions **Pio_RTC_DayOfWeek_Read**

Platform RASPBIAN on RASPBERRY PI

Parameter: "value"

"value" must be inside the range 1..7.

Return value

+1 for SUCCESS : the write operation is successful.
-1 for error : the write operation is not successful.

Example codes

Ex.1 : Write ' 25 April 2014 Friday' to RTC.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int RTC_year;
    int RTC_month;
    int RTC_dayofmonth;
    int RTC_dayofweek;
```

```
        RTC_year = 14;
        RTC_month = 4;
        RTC_dayofmonth = 25;
        RTC_dayofweek = 5;

        Pio_RTC_Year_Write(RTC_year);
        Pio_RTC_Month_Write(RTC_month);
        Pio_RTC_DayOfMonth_Write(RTC_dayofmonth);
        Pio_RTC_DayOfWeek_Write(RTC_dayofweek);

        while (1)
        {
            // YOUR CODE
        }
    }
```

Pio_RTC_DayOfWeek_Read

Declaration `int Pio_RTC_DayOfWeek_Read(void)`

Description Reads the value of the actual day of week from RTC.

Parameter Type/Description
none

Return value integer number – the current day of week.

Related functions **Pio_RTC_DayOfWeek_Write**

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful.
-1 for error : the read operation is not successful.

Example codes

Ex.1 : Read current day of week from RTC.

```
    int main (void)
    {
        Pio_SYSTEM_Init();

        int RTC_dayofmonth;

        RTC_dayofweek = Pio_RTC_DayOfWeek_Read();
        printf ("The current day of week is : %i\n", RTC_dayofweek);

        while (1)
        {
            // YOUR CODE
        }
    }
```

Pio_RTC_Hour_Write

Declaration `int Pio_RTC_Hour_Write(int value)`

Description Sets the hour of clock on the RTC.

Parameter Type/Description
value integer number - the value of hour on RTC.

Return value integer number – success or error code.

Related functions **Pio_RTC_Hour_Read**

Platform RASPBIAN on RASPBERRY PI

Parameter: “value”

“value” must be inside the range 0..24.

Return value

+1 for SUCCESS : the write operation is successful.
-1 for error : the write operation is not successful.

Example codes

Ex.1 : Write ’25 April 2014 Friday, 15 in the afternoon’ to RTC.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int RTC_year;
    int RTC_month;
    int RTC_dayofmonth;
    int RTC_dayofweek;
    int RTC_hour;

    RTC_year = 14;
    RTC_month = 4;
    RTC_dayofmonth = 25;
    RTC_dayofweek = 5;
    RTC_hour = 15;

    Pio_RTC_Year_Write(RTC_year);
    Pio_RTC_Month_Write(RTC_month);
    Pio_RTC_DayOfMonth_Write(RTC_dayofmonth);
    Pio_RTC_DayOfWeek_Write(RTC_dayofweek);
    Pio_RTC_Hour_Write(RTC_hour);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_RTC_Hour_Read

Declaration `int Pio_RTC_Hour_Read(void)`

Description Reads current hour from RTC.

Parameter Type/Description
none

Return value integer number – the current hour.

Related functions **Pio_RTC_Hour_Write**

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful.
-1 for error : the read operation is not successful.

Example codes

Ex.1 : Read current hour from RTC.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int RTC_hour;

    RTC_hour = Pio_RTC_Hour_Read();
    printf ("The current hour is : %i\n", RTC_hour);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_RTC_Minute_Write

Declaration `int Pio_RTC_Minute_Write(int value)`

Description Set the minute of clock on the RTC.

Parameter Type/Description
value integer number - the value of minute on RTC.

Return value integer number – success or error code.

Related functions **Pio_RTC_Minute_Read**

Platform RASPBIAN on RASPBERRY PI

Parameter: "value"

“value” must be inside the range 0..59.

Return value

+1 for SUCCESS : the write operation is successful.
-1 for error : the write operation is not successful.

Example codes

Ex.1 : Write ’25 April 2014 Friday, 15:20’ to RTC.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int RTC_year;
    int RTC_month;
    int RTC_dayofmonth;
    int RTC_dayofweek;
    int RTC_hour;
    int RTC_minute;

    RTC_year = 14;
    RTC_month = 4;
    RTC_dayofmonth = 25;
    RTC_dayofweek = 5;
    RTC_hour = 15;
    RTC_minute = 20;

    Pio_RTC_Year_Write(RTC_year);
    Pio_RTC_Month_Write(RTC_month);
    Pio_RTC_DayOfMonth_Write(RTC_dayofmonth);
    Pio_RTC_DayOfWeek_Write(RTC_dayofweek);
    Pio_RTC_Hour_Write(RTC_hour);
    Pio_RTC_Minute_Write(RTC_minute);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_RTC_Minute_Read

Declaration `int Pio_RTC_Minute_Read(void)`

Description Reads current minute from RTC.

Parameter Type/Description
none

Return value integer number – the current minute.

Related functions **`Pio_RTC_Minute_Write`**

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful.
-1 for error : the read operation is not successful.

Example codes

Ex.1 : Read current minute from RTC.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int RTC_minute;

    RTC_minute = Pio_RTC_Minute_Read();
    printf ("The current minute is : %i\n", RTC_minute);

    while (1)
    {
        // YOUR CODE
    }
}
```

Pio_RTC_Second_Write

Declaration	`int Pio_RTC_Second_Write(int value)`
Description	Sets the second of clock on the RTC.
Parameter	Type/Description
value	integer number - the value of second on RTC.
Return value	integer number – success or error code.
Related functions	**Pio_RTC_Second_Read**
Platform	RASPBIAN on RASPBERRY PI

Parameter: "value"

"value" must be inside the range 0..59.

Return value

+1 for SUCCESS : the write operation is successful.
-1 for error : the write operation is not successful.

Example codes

Ex.1 : Write '25 April 2014 Friday, 15:20:45' to RTC.

```
int main (void)
{
    Pio_SYSTEM_Init();
```

```
    int RTC_year;
    int RTC_month;
    int RTC_dayofmonth;
    int RTC_dayofweek;
    int RTC_hour;
    int RTC_minute;
    int RTC_second;

    RTC_year = 14;
    RTC_month = 4;
    RTC_dayofmonth = 25;
    RTC_dayofweek = 5;
    RTC_hour = 15;
    RTC_minute = 20;
    RTC_second = 45;

    Pio_RTC_Year_Write(RTC_year);
    Pio_RTC_Month_Write(RTC_month);
    Pio_RTC_DayOfMonth_Write(RTC_dayofmonth);
    Pio_RTC_DayOfWeek_Write(RTC_dayofweek);
    Pio_RTC_Hour_Write(RTC_hour);
    Pio_RTC_Minute_Write(RTC_minute);
    Pio_RTC_Second_Write(RTC_second);

    Pio_RTC_Start();

    while (1)
    {
        delay(1000);
        printf ("The current second is : %i\n", RTC_second);
        // YOUR CODE
    }
}
```

Pio_RTC_Second_Read

Declaration `int Pio_RTC_Second_Read(void)`

Description Reads current second from RTC.

Parameter Type/Description
none

Return value integer number - the current second.

Related functions **Pio_RTC_Second_Write**

Platform RASPBIAN on RASPBERRY PI

Return value

The requested value for SUCCESS : the read operation is successful.
-1 for error : the read operation is not successful.

Example codes

Ex.1 : Read current second from RTC.

```
int main (void)
{
    Pio_SYSTEM_Init();

    int RTC_second;

    RTC_second = Pio_RTC_Second_Read();
    printf (“The current second is : %i\n”, RTC_second);

    while (1)
    {
        // YOUR CODE
    }
}
```

Using PROGRAMMING LANGUAGES to write your code

- C
- PYTHON
- JAVA

PROGRAMMING LANGUAGES General notes

Depending of programming language chosen to write code you will need to use the libraries 'libpio.so' or 'libpio.jar', available for download in our site www.intellisys.it.

- How to do to use library 'libpio' - :

- Create a directory of project for the code you want write.
- Copy the file 'libpio.so' or 'libpio.jar' inside that directory of the project.
- Write your code in one of the programming language available.
- Compile your project following the instructions below.

Programming in C code

To write your own code to have access to the PIO interface you must use the library '**libpio.so**' and the '**libpio.h**' file and these 2 files must be inside your project directory. The library 'libpio.so' is an indispensable tool for any programmer who wants use PIO interface. It is a pre-existing code that is compiled and ready for your use. The 'libpio.so' is a shared library written in PIC (position independent code) code that works no matter where in memory it is placed. This is because several different programs can all, at the same time, use an instance of 'libpio.so', so the library cannot store anything at a fixed address.
Warning: any program that uses the library '**libpio.so**' must be run as '**super user**'. For example, to run a program named "my_prog", you must use the command **'sudo. /my_prog'**.

Example codes

Ex.1 : Write your program code to blinking the red led every 1 second. Call your program '**piored.c**'. Assuming that the username is 'pi', save 'piored.c' file in the directory **'/home/pi/my_project'**. If the username is different replace 'pi' with your 'username'.

File 'piored.c' :

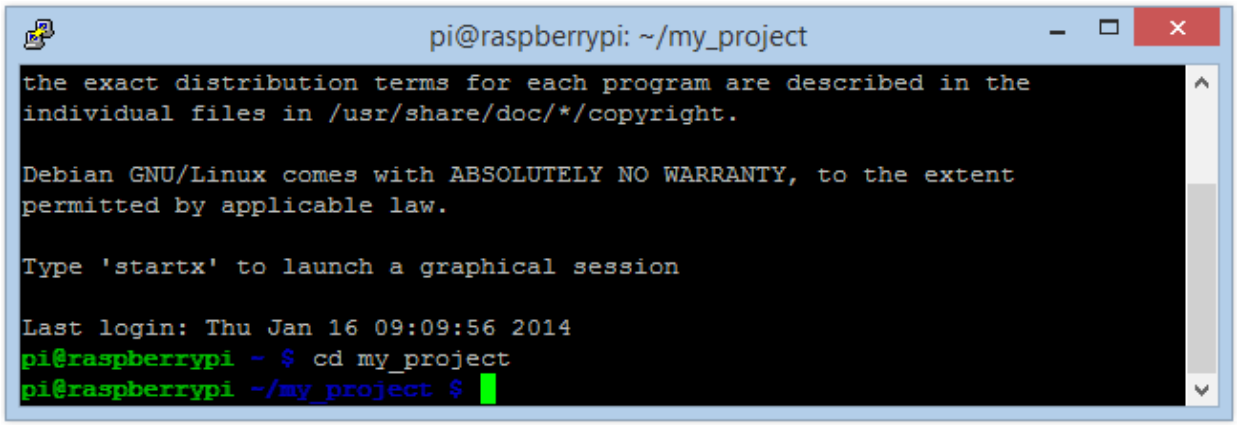
```
#include "libpio.h"  // This is an important line you must insert in your C code!
#include <unistd.h>

int main (void)
{
    Pio_SYSTEM_Init();
    while (1)
    {
        Pio_LED_Command_Write (3,1);
        sleep(1);
        Pio_LED_Command_Write (3,0);
        sleep(1);
    }
}
```

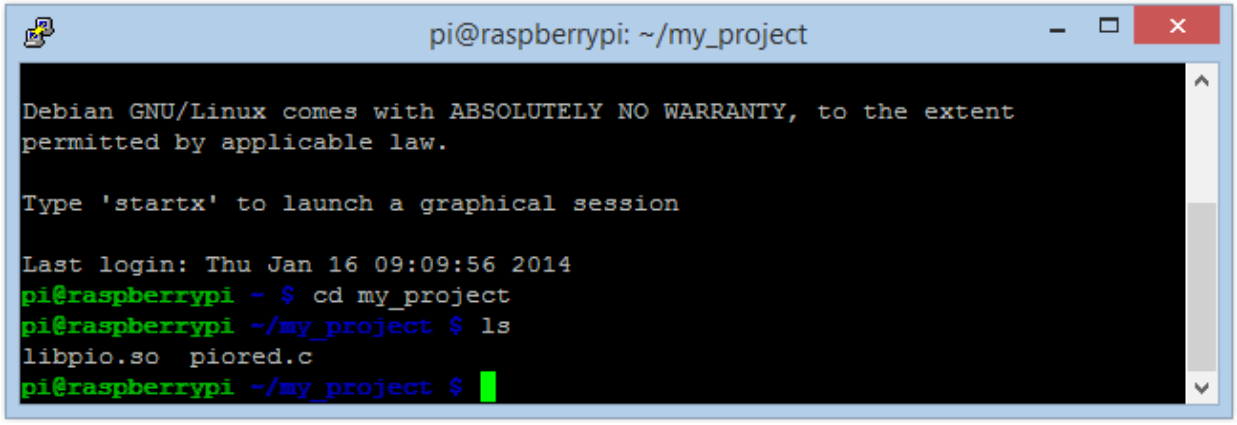
Compile now the program 'piored.c' following the next instructions:

Case 1 : compile 'piored.c' **inside** 'my_project' folder.

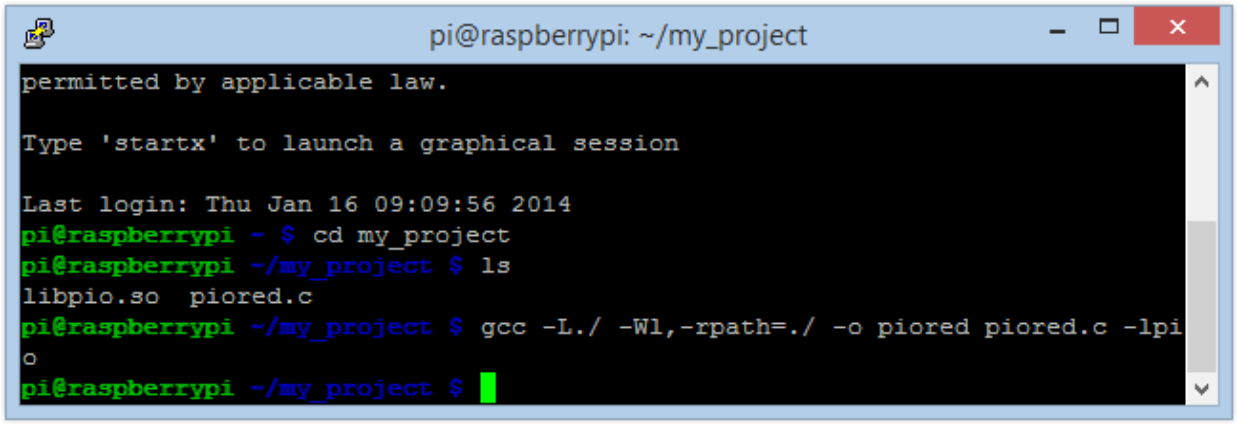
- Open an instance of terminal window and type :
  cd my_project and press enter



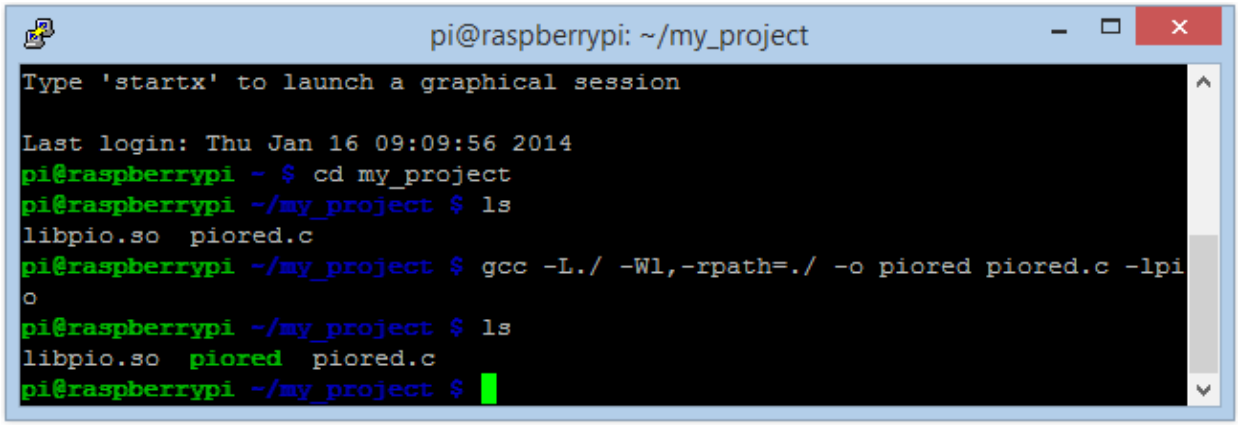
- Verify that in the folder ‘my_project’ you find ‘piored.c’ and ‘libpio.so’ files typing
  ls and press enter



- Type :
  gcc -L./ -Wl,-rpath=./ -o piored piored.c -lpio and press enter



- Verify that in the folder ‘my_project’ you find ‘piored.c’, ‘libpio.so’ files and ‘piored’ program typing
  ls and press enter



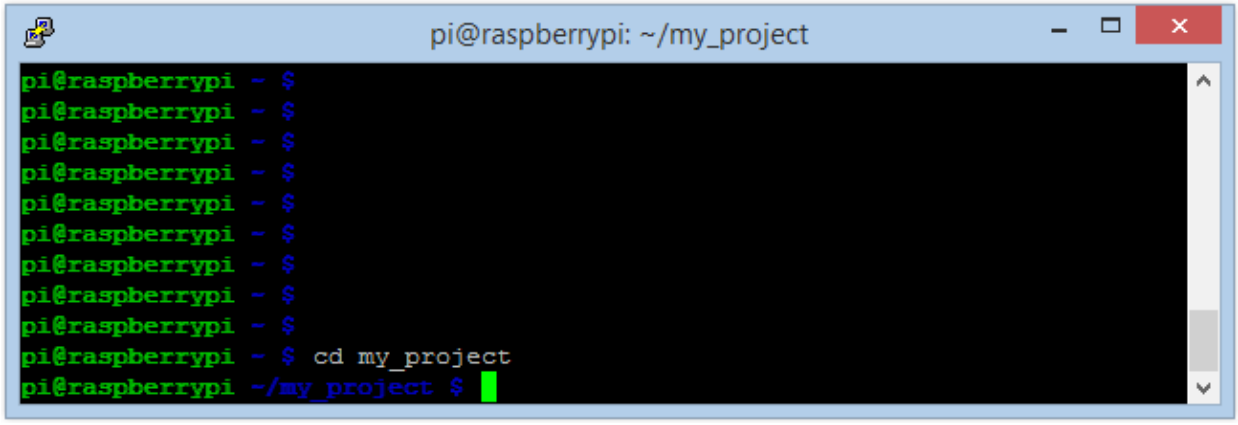
- Run the program typing:

sudo ./piored and press enter

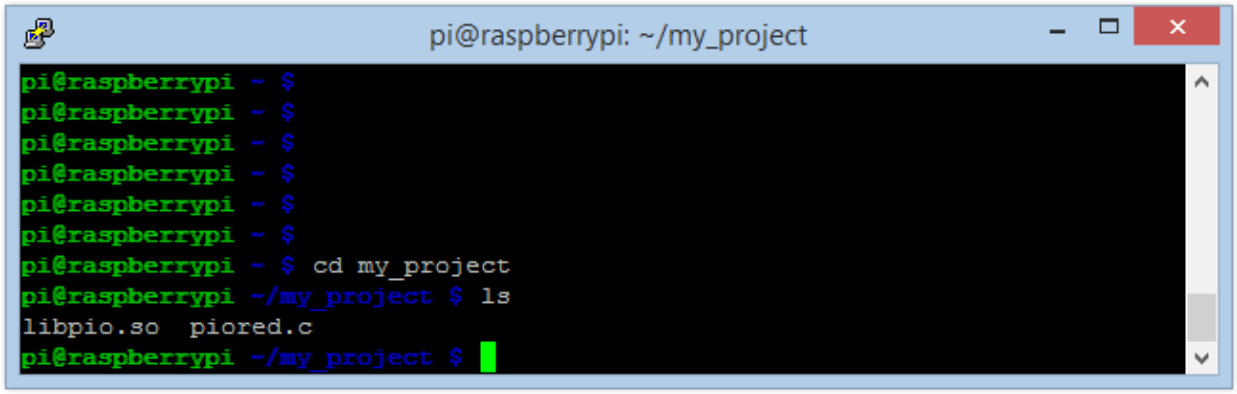
- You will see the red led which turning ON and OFF every 1 second.

Case 2 : compile ‘piored.c’ **outside** ‘my_project’ folder.

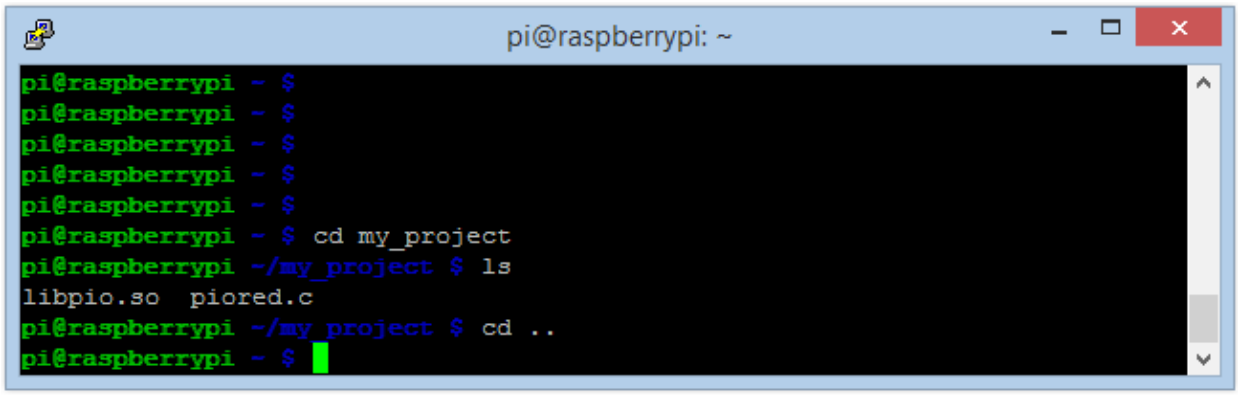
- Open an instance of terminal window and type :
  `cd my_project` and press enter



- Verify that in the folder ‘my_project’ you find ‘piored.c’ and ‘libpio.so’ files typing
  `ls` and press enter

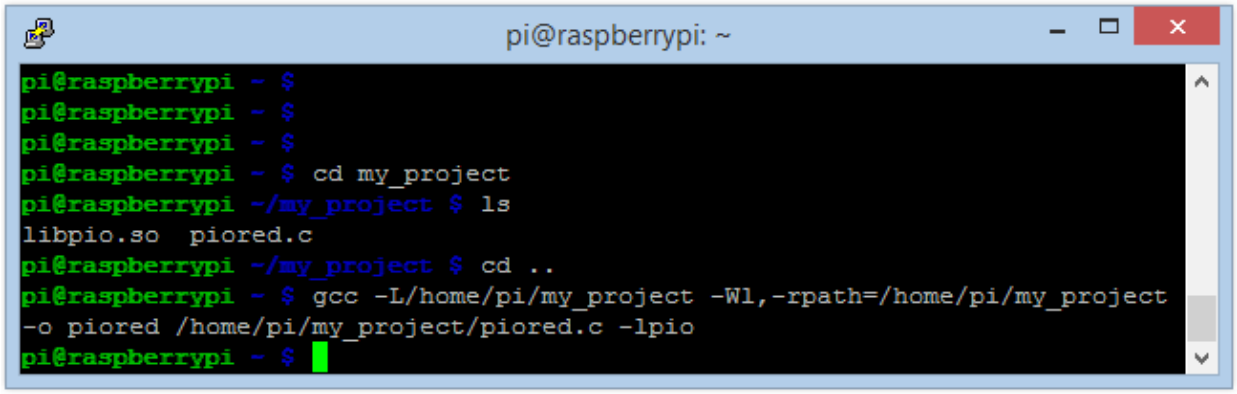


- Type :
  `cd ..` and press enter



- Compile ‘piored.c’ typing:

gcc -L/home/pi/my_project -Wl,-rpath=/home/pi/my_project -o piored /home/pi/my_project/piored.c -lpio and press enter



- Run the program typing:
  sudo ./piored and press enter

- You will see the red led which turning ON and OFF every 1 second.

Ex.2: Write your program code to blinking the first green led every 2 seconds. Call your program **'piogreen.c'**. Assuming that the username is 'pi', save 'piogreen.c' file in the directory **'/home/pi/my_project'**. If the username is different replace 'pi' with your 'username'.

File 'piogreen.c' :

```
#include "libpio.h"  // This is an important line you must insert in your C code!
#include <unistd.h>

int main (void)
{
    Pio_SYSTEM_Init();

    while (1)
    {
        Pio_LED_Command_Write (1,1);
        sleep(2);
        Pio_LED_Command_Write (1,0);
        sleep(2);
    }
}
```

Compile now the program 'piogreen.c' following the next instructions:

- Open an instance of terminal window and type :
  `cd my_project` and press enter
- Verify that in the folder 'my_project' you find 'piogreen.c' and 'libpio.so' files typing
  `ls` and press enter
- Type :
  `gcc -L./ -Wl,-rpath=./ -o piogreen piogreen.c -lpio` and press enter
- Run the program typing:
  `sudo ./piogreen` and press enter
- You will see the green led which turning ON and OFF every 2 seconds.

Ex.3 : Write your program code to turn On and OFF the first digital output relay every 3 seconds. Call your program **'piorelay.c'**. Assuming that the username is 'pi', save 'piorelay.c' file in the directory **'/home/pi/my_project'**. If the username is different replace 'pi' with your 'username'.

File 'piorelay.c' :

```
#include "libpio.h"  // This is an important line you must insert in your C code!
#include <unistd.h>

int main (void)
{
    Pio_SYSTEM_Init();

    while (1)
    {
        Pio_DO_Command_Write (1,1);
        sleep(3);
        Pio_DO_Command_Write (1,0);
        sleep(3);
    }
}
```

Compile now the program 'piorelay.c' following the next instructions:

- Open an instance of terminal window and type :
  `cd my_project` and press enter
- Verify that in the directory 'my_project' you find 'piorelay.c' and 'libpio.so' files typing
  `ls` and press enter
- Type :
  `gcc –L./ -Wl,-rpath=./ -o piorelay piorelay.c –lpio` and press enter
- Run the program typing:
  `sudo ./piorelay` and press enter
- You will see the first relay which turning ON and OFF every 3 seconds.

Ex.4 : Test the shared library 'libpio.so' running simultaneously the programs 'piored', 'piogreen' and 'piorelay'.

To do this open three instances of terminal window and in each one type the command to run one of the 3 programs, i.e. '**`sudo ./piored`**' in the first window, '**`sudo ./piogreen`**' in the second window and '**`sudo ./piorelay`**' in the third window.

Warning: the examples above refer to different processes that using different functions and resources. If you want to perform different processes (task) that use the same function you must resort to particular programming techniques (semaphores, mutexes) i.e. to the concurrent programming. This method is out of scope for this manual.

Programming in PYTHON code

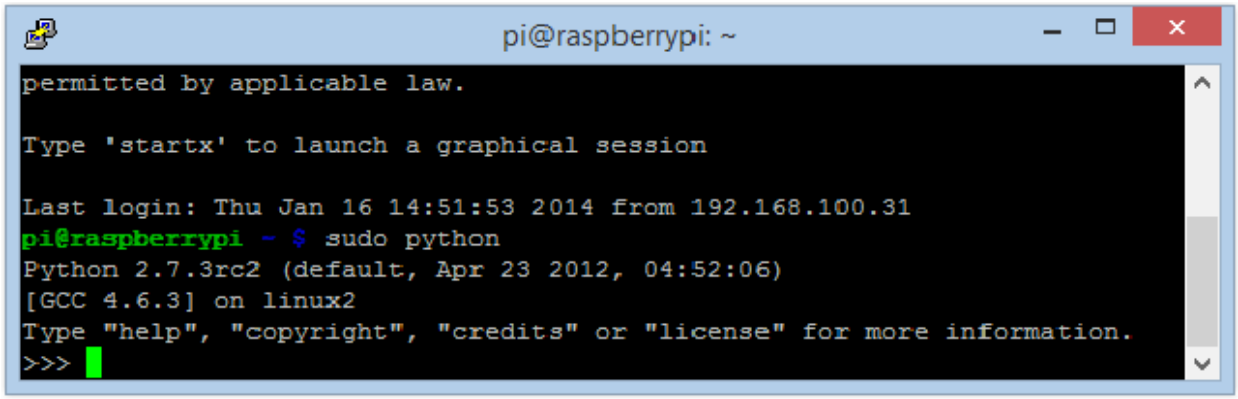
To write your own code to have access to the PIO interface you must use the library '**libpio.so**'.
The library 'libpio.so' is an indispensable tool for any programmer who wants use PIO interface. It is a pre-existing code that is compiled and ready for your use. The 'libpio.so' is a shared library written in PIC (position independent code) code that works no matter where in memory it is placed. This is because several different programs can all, at the same time, use an instance of 'libpio.so', so the library cannot store anything at a fixed address.
Warning: any program that uses the library '**libpio.so**' must be run as '**super user**'. For example, to run a program named "my_prog", you must use the command '**`sudo. /my_prog`**'.

Example codes

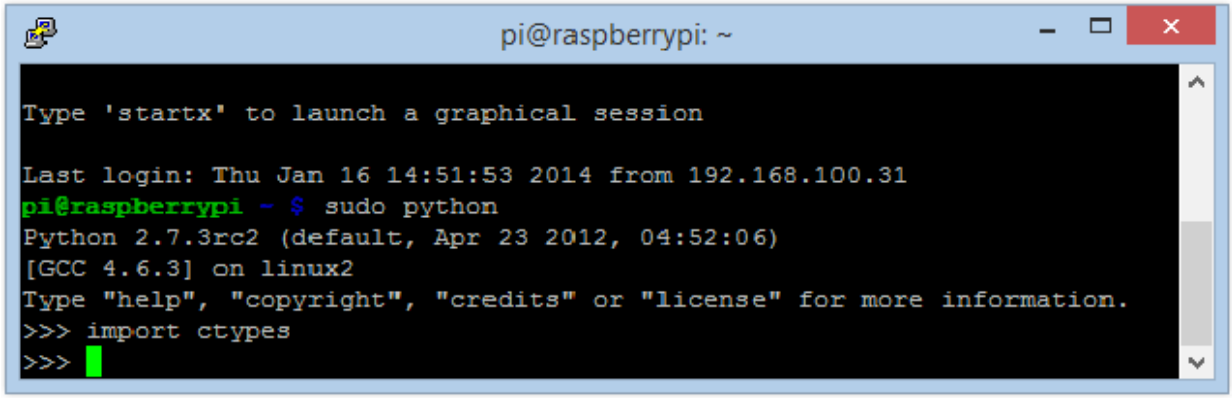
Ex.1 : Turn ON the red Led on PIO interface and then turn it OFF.

Follow the next instructions:

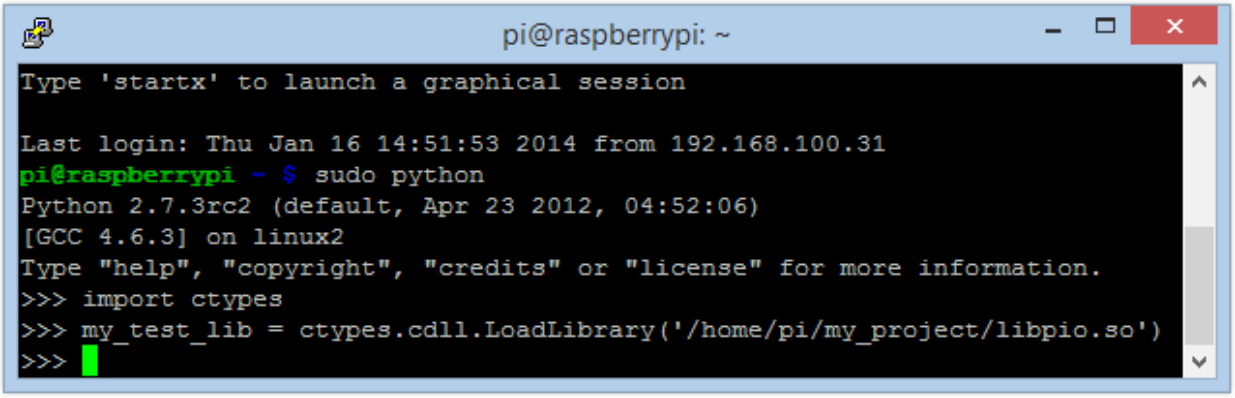
- Open an instance of terminal window and type :
  `sudo python` and press enter



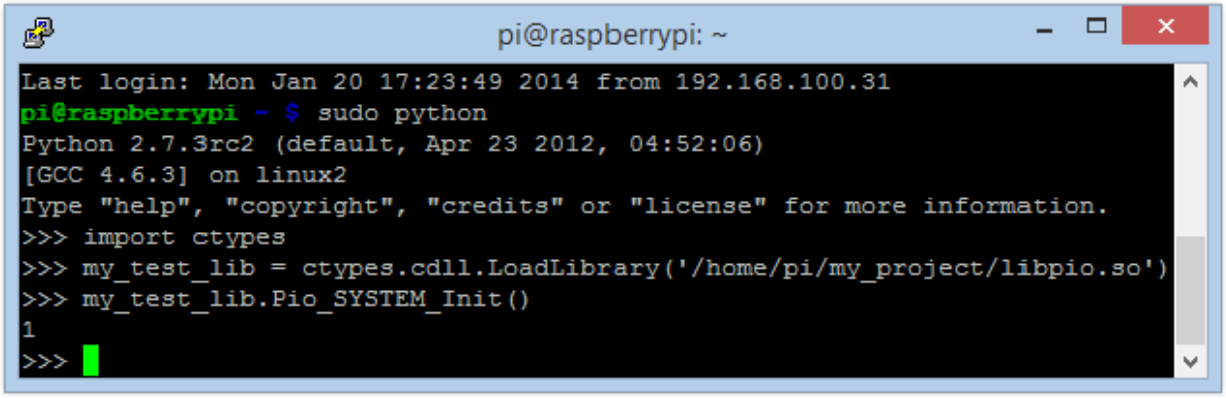
- type:
  `import ctypes` and press enter



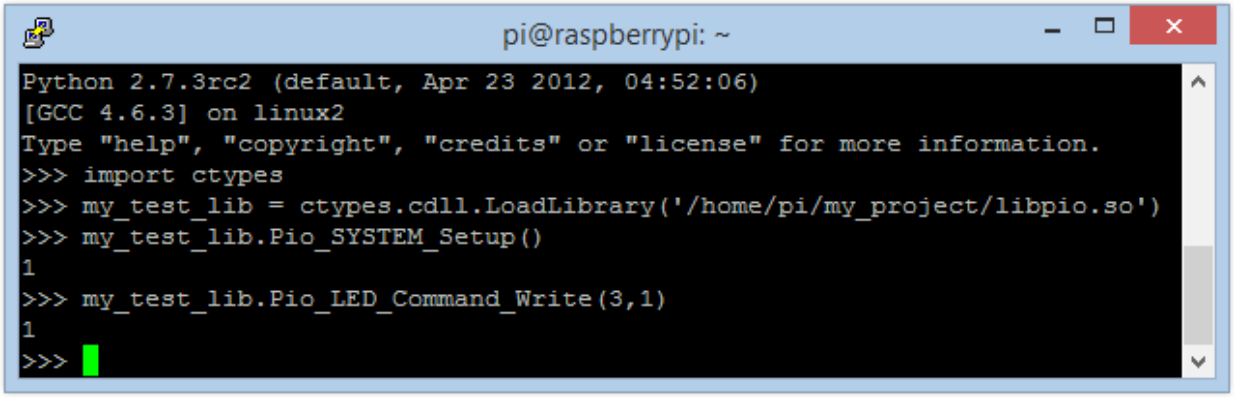
- type:
  `my_test_lib = ctypes.cdll.LoadLibrary('/home/pi/my_project/libpio.so')`
  and press enter



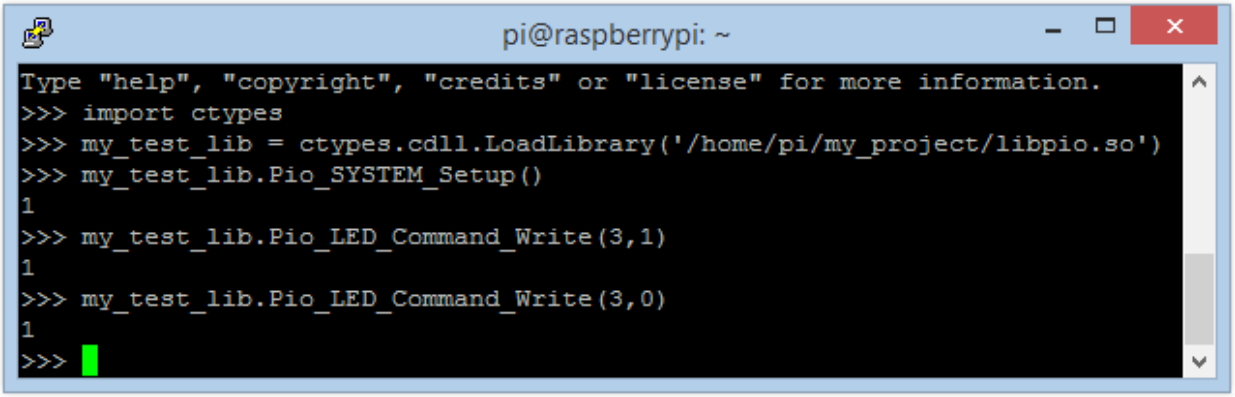
- type:
  my_test_lib.Pio_SYSTEM_Init() and press enter



- type:
  my_test_lib.Pio_LED_Command_Write(3,1) and press enter



- type:
  my_test_lib.Pio_LED_Command_Write(3,0) and press enter



With the last two commands the red led is ON and after OFF.

Floating points values using 'libpio.so' shared library

For each foreign function, coming from 'libpio.so', it is very important declare a 'ctypes' type to the result and to the arguments of the function. To do that you can use two keywords: 'restype' and 'argtypes'. Only if the 'ctypes' type is '**int**' you can omit declaration.

restype:

Assign ctypes type to specify the result type of the foreign function.

argtypes:

Assign 'ctypes' type to specify the argument types that the function accepts.

Example

>>> my_test_lib.Pio_UI_Value_Read.restype = ctypes.c_float
>>> my_test_lib.Pio_UI_Value_Read.argtypes = [ctypes.c_int]

If the number of arguments are more than one, write a comma between arguments, like in the next example:
>>> my_test_lib.Pio_UI_BottomRange_Write.argtypes = [ctypes.c_int, ctypes.c_float]

For more details see Python language reference guide.

Programming in JAVA code

To write your own code to have access to the PIO interface you must use the library '**libpio.jar**'.
The library 'libpio.jar' is an indispensable tool for any programmer who wants use PIO interface. It is a pre-existing code that is compiled and ready for your use. The 'libpio.jar' is a porting of 'libpio.so' library for use in C (see Programming in C code section for more details).
Warning: any program that uses the library '**libpio.jar**' must be run as '**super user**'.

Example codes

Ex.1 : Write your program code to turn On and OFF the first digital output relay every 3 seconds. Call your class '**PioRelay.java**'. Assuming that the username is 'pi', save 'PioRelay.java' file in the directory '**/home/pi/java_project**'. If the username is different replace 'pi' with your 'username'.

File 'PioRelay.java':

```
import it.intellisys.pio.LibPio;

public class PioRelay {

	public static void main(String[] args) {
		LibPio libPio = new LibPio();
		libPio.Pio_SYSTEM_Init();

		while (true) {
			pioLib.Pio_DO_Command_Write(1, 1);
			try { Thread.sleep(3000);}
			catch(InterruptedException ex) { return; }

			pioLib.Pio_DO_Command_Write(1, 0);
			try { Thread.sleep(3000);}
			catch(InterruptedException ex) { return; }
		}
	}
}
```

Warning: Remember to import the package of LibPio library class using:

```
import it.intellisys.pio.LibPio;
```

Compile now the class 'PioRelay.java' following the next instructions:

- Open an instance of terminal window and type :
  `cd java_project` and press enter
- Verify that in the folder 'java_project' you find 'PioRelay.java' and 'libpio.jar' files typing
  `ls` and press enter
- Compile in byte code the class 'PioRelay.java' typing:
  `javac -cp .:libpio.jar PioRelay.java` and press enter
- Verify that in the folder 'java_project' you find 'PioRelay.java', 'libpio.jar' and 'PioRelay.class' files typing
  `ls` and press enter
- Run the program typing:
  `sudo java -cp .:libpio.jar PioRelay` and press enter
- You will see the first relay which turning ON and OFF every 3 seconds.