Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN

Trabajo Fin de Grado

EQUIPO DE MONITORIZACIÓN DE MAGNITUDES ELÉCTRICAS BASADO EN RASPBERRY PI

Alumno: **Sergiy Yaremchuk**

Tutor: Ángel Gaspar González Rodríguez

Dpto: Ingeniería Electrónica y Automática

Junio, 2017

Contenido

1. INTRODUCCIÓN GENERAL

1.1. Estructura del trabajo

En este trabajo se describe el diseño, selección, construcción y programación de un equipo destinado a la medición de las variables características de una instalación eléctrica y su visualización en cualquier dispositivo con acceso a internet.

El trabajo se divide en un índice, siete apartados de desarrollo, cinco apartados de anexos, uno de glosario y uno de bibliografía.

En el primer apartado, se da una visión superficial sobre el proyecto destacando la importancia de la metrología a lo largo de la historia y definiendo los objetivos del proyecto sin entrar en cuestiones técnicas del mismo.

En el segundo apartado, se enumeran los materiales del proyecto con ilustraciones y una introducción de la funcionalidad y características de cada uno de cara al proyecto.

En el tercer apartado, se define cómo van conectados los diferentes componentes del proyecto y el motivo de dicho conexionado.

En el cuarto apartado, se explica la configuración de los componentes para su adecuado funcionamiento como conjunto. Instalación de módulos, configuración de los dispositivos usados, configuración de dispositivos auxiliares, etc.

En el quinto apartado, se explica el funcionamiento del dispositivo. Se divide en varias fases en las que los datos se toman, se guardan, y se envían para la visualización final desde un dispositivo remoto. Se incluyen también varios apartados de gestión de errores.

En el sexto apartado, se calcula el presupuesto aproximado del proyecto a partir de los precios de cada componente.

En el séptimo apartado, se proponen una serie de mejoras para el proyecto. Mejoras tanto de hardware como de software (código principal).

En el noveno apartado se exponen algunas conclusiones sobre el trabajo realizado.

El Anexo I contiene el manual de uso del Circutor.

El Anexo II contiene las características del módulo MinimalModbus.

El Anexo III contiene las características del módulo xml.dom.minidom.

El Anexo IV contiene las características del módulo ftplib.

En el Anexo V se encuentran los códigos de las partes del proyecto que han sido codificadas (Raspberry, dispositivo Android, servidor remoto). Se han utilizado los lenguajes Python, Java y PHP.

El penúltimo apartado es un glosario con algunas definiciones de términos y acrónimos.

El último apartado es de bibliografía. Contiene todas las citas bibliográficas citadas a lo largo del trabajo. Todas las citas han sido sacadas de páginas web. La fecha de las citas es la fecha de último acceso.

1.2. Introducción

La palabra *"medir"* proviene del término en latín *"metïri"*, según la RAE significa comparar una cantidad con su respectiva unidad, con el fin de averiguar cuántas veces la segunda está contenida en la primera. (RAE, 2017)

La operación de medir, más antigua incluso que el propio término, se ha realizado desde el año 5000a.C, cuando el hombre comenzó a utilizar su propio cuerpo como unidad de medida (unidades antropomórficas). En la ***Ilustración 1.2.1*** se muestran las principales unidades antropomórficas para medir longitudes utilizadas en épocas anteriores. (Castillo, 2017)

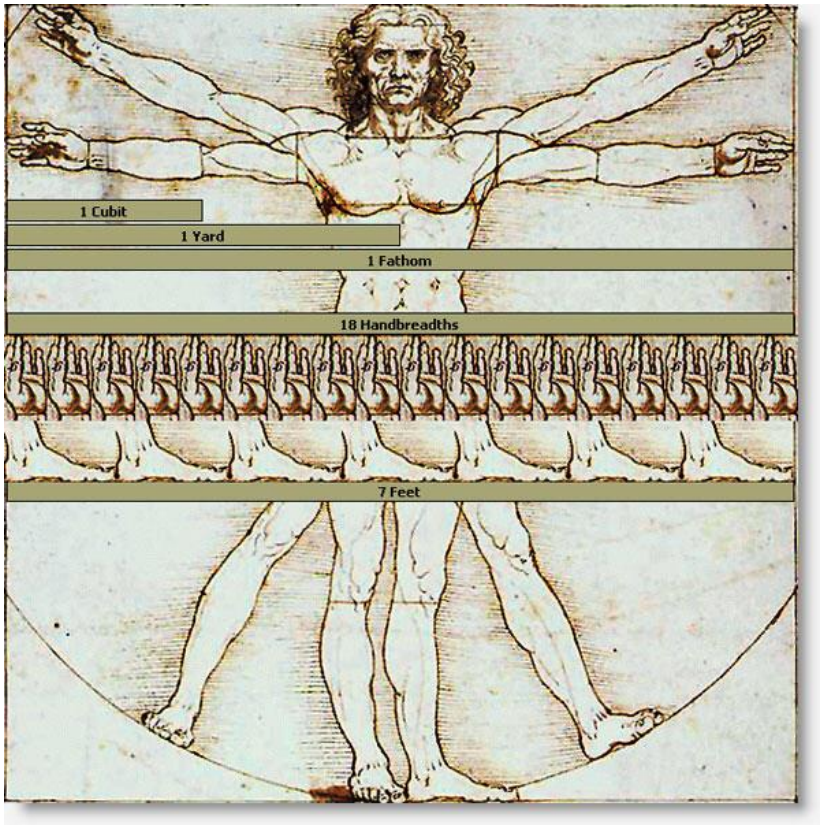


Ilustración 1.2.1

Sin embargo, todas las unidades de medida resultaban imperfectas, ya que variaban de según el individuo y el lugar de medida. Lo cual creaba dificultades en las primeras relaciones comerciales entre las personas. Con el paso del tiempo, el desarrollo de la industria y la ciencia fueron obligando al desarrollo de las medidas ya que estas iban tomando importancia en el ámbito cotidiano, en la producción, distribución, industria, etc.

A lo largo de la historia, muchos descubrimientos relacionados con la medición de magnitudes han supuesto un avance enorme en otros ámbitos. Algunos de estos descubrimientos fueron el de Arquímedes, que descubrió cómo medir el volumen de un cuerpo irregular sumergiéndolo en agua o el de Eratóstenes, que halló la forma de medir la circunferencia terrestre de forma muy aproximada observando la sombra que producía una vara en dos ciudades diferentes.

La metrología ha crecido significativamente hasta nuestros días. Actualmente, poseemos una gran cantidad de unidades de medida para cualquier magnitud medible e incluso varias unidades de medida por magnitud. La medición es una de las operaciones más fundamentales en el campo de la ciencia y sobre todo de la ingeniería.

Ante esta creciente necesidad de medición se han creado numerosos instrumentos de medida para casi cualquier magnitud, los cuales comenzaron siendo mecánicos y con la aparición de la electrónica se han ido transformando en electrónicos ganando en precisión y fiabilidad y reduciendo errores humanos.

Como consecuencia y teniendo en cuenta que es conveniente tener los datos en formato digital para un procesamiento y visualización más dinámicos, conocer tanto los métodos para extraer magnitudes con aparatos electrónicos como el funcionamiento de dichos aparatos es de gran utilidad en el ámbito de la ingeniería actual.

1.3. Objetivo del proyecto

La principal finalidad del proyecto es el control del consumo energético de una instalación eléctrica. Se ha de poder implementar en instalaciones eléctricas de todo tipo de edificios (domésticos, industriales, etc.).

Hoy en día, muchos de los contadores de luz siguen siendo analógicos (***Ilustración 1.3.1***) y, en cambio de algunos digitales, no disponen de ninguna herramienta para extraer los datos de consumo. En este proyecto se podrán extraer las medidas de un contador analógico y también será compatible con contadores digitales en caso de que no dispongan de un método para extraer las medidas o se ponga en duda su fiabilidad.

Ilustración 1.3.1

La visualización de los datos se realizará online y cada usuario poseerá una clave para consultar los datos de su instalación desde una web o desde una aplicación Android. La actualización de los datos medidos se realizará con periodos de tiempo de dos minutos.

Aparte del control de consumo se plantea como objetivo monitorizar otras variables como energía parcial o energías generada activa y reactiva interesantes para el control de una instalación industrial con maquinaria más compleja. En este proyecto, las variables adicionales son calculadas automáticamente por el aparato de medida del que se extraen, en caso de que no fuera así, se podrían calcular a partir de las variables básicas.

2. MATERIALES

El proyecto ha sido adaptado a los materiales disponibles. Por tanto, algunos de los materiales elegidos no son los más eficientes en cuanto a tamaño, consumo y relación calidad/precio.

Listado de materiales:

1. *Raspberry Pi 1 Model B+*
2. *Circutor CVM-1D*
3. *Cable Ethernet (Opcional)*
4. *Dos cables trenzados para comunicación RS-485*
5. *Convertidor RS-485 to USB*
6. *Reloj RTC (DS3231)*
7. *USB Dongle WI-FI (Opcional)*
8. *Fuente de alimentación 220/5V 600mA*
9. *Cable HDMI*
10. *Ratón y Teclado USB*
11. *Pantalla de PC oTV con entrada HDMI*

2.1. Raspberry Pi 1 Model B+

Una Raspberry Pi es un computador de placa reducida de bajo coste desarrollado por una fundación de Reino Unido. Es un dispositivo cuyas especificaciones, diseños y esquemas son de acceso público y siguen, en cierta forma, la filosofía del software libre. (RASPBERRY PI FOUNDATION, 2017)

Este dispositivo tiene grandes utilidades tanto en el campo profesional como en el particular. Puede ser utilizado para casi cualquier tarea de control que no requiera gran esfuerzo de computación, por ejemplo, en la electrónica del hogar. En proyectos como equipos de música, estaciones multimedia o medidores, la Raspberry Pi se desenvuelve con suficiencia, mientras que para tareas más complejas como controlar varios procesos complejos o tareas que necesiten mayor esfuerzo de computación, la Raspberry puede resultar lenta e inestable. Por ello, es más recomendable su uso en el ámbito académico o doméstico.

Su principal desventaja es la cantidad de fallos que se producen por razones como incompatibilidad de librerías y dispositivos, lecturas de código erróneas, fallos del sistema basado en linux, etc.

La Raspberry Pi puede ejecutar varios lenguajes de programación. El más usado es Python ya que tiene una gran cantidad de librerías compatibles con Raspberry y es un lenguaje muy claro. Este será el lenguaje utilizado en este proyecto.

El modelo utilizado es la versión final de la primera Raspberry Pi (ver ***Iustración 2.1.1***) cuyas características, aunque sean algo limitadas en comparación con las nuevas versiones, para este proyecto serán suficientes.

Ilustración 2.1.1

Tabla de características:

SISTEMA EN CHIP (SOC)	Broadcom BCM2835 (CPU + GPU + DSP + SDRAM + puerto USB)
CPU	ARM 1176JZF-S a 700 MHz (familia ARM11)
Juego de instrucciones	RISC de 32 bits
GPU	Broadcom VideoCore IV, OpenGL ES 2.0, MPEG-2 y VC-1 (con licencia), 1080p30 H.264/MPEG-4 AVC
Memoria (SDRAM)	512 MiB (compartidos con la GPU)
Puertos USB 2.0	4
Entradas de vídeo	Conector MIPI CSI que permite instalar un módulo de cámara desarrollado por la RPF
Salidas de vídeo	Conector RCA (PAL y NTSC), HDMI (rev1.3 y 1.4), Interfaz DSI para panel LCD
Salidas de audio	Conector de 3.5 mm, HDMI

Almacenamiento integrado	MicroSD
Conectividad de red	10/100 Ethernet (RJ-45) via hub USB
Periféricos de bajo nivel	8 x GPIO, SPI, I²C, UART (40 PINS)
Consumo energético	600 mA, (3.0 W)
Fuente de alimentación	5 V vía Micro USB o GPIO header
Dimensiones	85.60mm × 53.98mm (3.370 × 2.125 inch)
Sistemas operativos soportados	GNU/Linux: Debian (Raspbian), Fedora (Pidora), Arch Linux (Arch Linux ARM), Slackware Linux, SUSE Linux Enterprise Server for ARM. RISC OS

Tabla 2.1.1

Diagrama de componentes: (MATT, 2017)

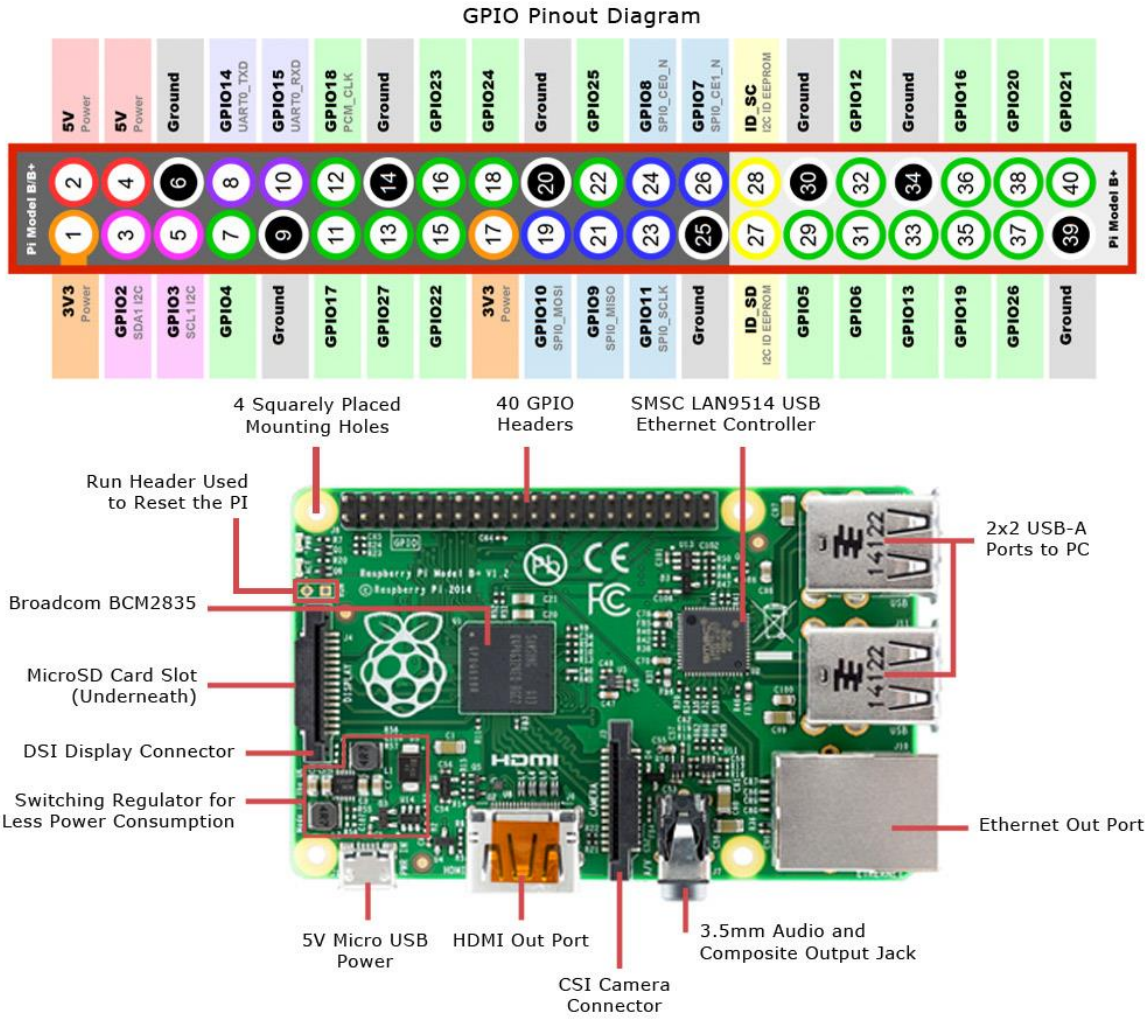


Ilustración 2.1.2

2.2. Circutor CVM-1D

Es un analizador de redes para circuitos monofásicos de hasta 32 A. Dispone de un display LCD de 6 dígitos con un sistema de pantallas rotativas, mostrando un total de 24 variables eléctricas de tipo instantáneo, máximo y mínimo. Se ha diseñado en una envolvente de tan sólo 1 módulo DIN (18mm), pudiendo instalar dicho analizador en cualquier cuadro eléctrico dado su reducido espacio. (CIRCUTOR, 2017)

Ilustración 2.2.1

Las principales características de este aparato son un puerto de comunicación RS-485 Modbus/RTU, salida de impulsos o alarma programable y medida en cuatro cuadrantes. Debido a su versatilidad se puede instalar en todo tipo de edificios: Residencias de estudiantes, hoteles, puertos deportivos, centros comerciales, edificios de alquiler de oficinas, campings, líneas domésticas e industriales.

Las variables medidas son: Tensión, Corriente, Potencia Activa, Potencia Reactiva (L/C), Potencia Aparente, Factor de Potencia, Energía Activa, Energía Reactiva, Contadores Parciales de Energía.

Para más información consultar **Anexo I.**

2.3. Cable Ethernet

Se utiliza para la comunicación de alta velocidad entre equipos de forma local (LAN). En este proyecto se ha utilizado para la comunicación entre Raspberry y un router. De esta forma, el dispositivo obtiene conexión a internet sin necesidad de conexión wifi.

También es posible manejar la Raspberry de forma remota por Wi-Fi (con USB Dongle Wi-Fi), pero para ello primero hay que configurarlo desde la propia Raspberry conectándola a la red local mediante el cable Ethernet o a una pantalla un teclado y ratón para configurarla desde su sistema operativo.

Ilustración 2.3.1

2.4. Dos cables trenzados para la comunicación RS-485

La comunicación RS-485 Está definida como un sistema de bus diferencial multipunto ideal para transmitir a altas velocidades sobre largas distancias (35 Mbit/s hasta 10 metros y 100 kbit/s en 1200 metros). Es ideal para transmitir a través de canales ruidosos, ya que el par trenzado reduce los ruidos que se inducen en la línea de transmisión. (Piña, 2017)

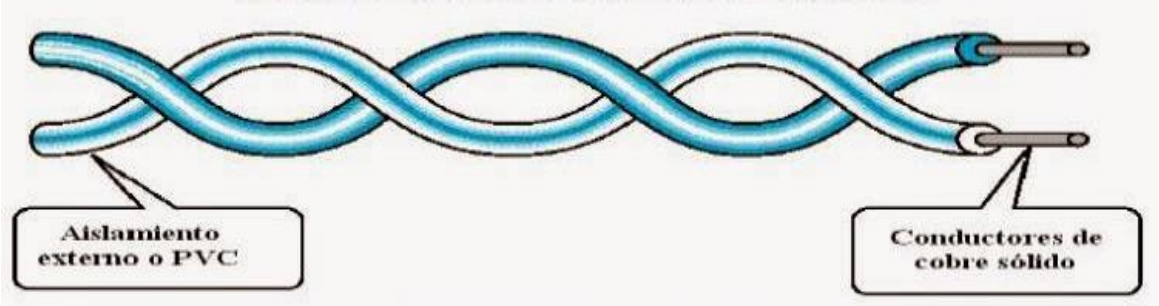


Ilustración 2.4.1

Dos alambres paralelos constituyen una antena simple. Cuando estos se trenzan, las ondas se cancelan, por lo que la interferencia producida en ambos también se cancela y se consigue una mejor transmisión de datos.

Características principales: La alimentación es única de +5V, permite conectar hasta 32 estaciones (ya existen interfaces que permiten conectar 256 estaciones) el bus tiene un rango de -7V a +12V.

2.5. Convertidor RS-485 to USB

Consiste en un conector con aspecto y tamaño similares a los de un pen drive, que sirve para cambiar el formato de la comunicación RS-485 a USB. Este dispositivo es necesario para poder conectar el Circutor a la Raspberry (que no dispone de entrada RS-485). (SODIAL(R), 2017)

Ilustración 2.5.1

Características:

No necesita una fuente de alimentación externa, ya que se alimenta por el puerto USB.

Puede ser utilizado tanto para lectura como para escritura.

Dimensiones: 18mm x 14mm x 60mm.

Peso aproximado: 8g.

Funciona en Windows XP, Vista, Windows 7, Windows 8 / 8.1, Linux, MacOS, WinCE5.0.

Velocidad: 75bps~115200bps, más de 6Mbps.

2.6. Reloj RTC

Cuando la Raspberry arranca, conecta con un servidor llamado NTP para consultar la hora. En ocasiones puede ocurrir que dicho servidor no esté disponible o que la Raspberry no tenga conexión a internet para poder acceder a él, por lo que se utiliza este reloj que debe ser configurado con la hora del sistema cuando es correcta. (Raspipc.es, 2017)

En este proyecto hará falta saber la hora exacta de la toma de las medidas para que los cálculos de consumo sean exactos, por lo que este reloj será útil ante posibles reinicios, caídas de la red o de servidor.

El modelo utilizado es el DS3231, que posee varias mejoras respecto a otras versiones como una mayor precisión. Posee oscilador interno incorporado con compensación de temperatura, que compensa la hora en función de la temperatura y de la edad del dispositivo. Con esta característica se obtiene una gran precisión en el rango de temperatura industrial -40°C a +85°. (Ventura, 2017)

Otras características:

Bajo consumo de energía (importante cuando está funcionando con una pila).

Libera de trabajo al sistema principal para que pueda dedicarse a tareas más críticas.

Algunas veces más preciso que otros métodos.

Se conecta en las patillas 1,3,5,7 tal y como se muestra en la ***Ilustración 2.6.1***

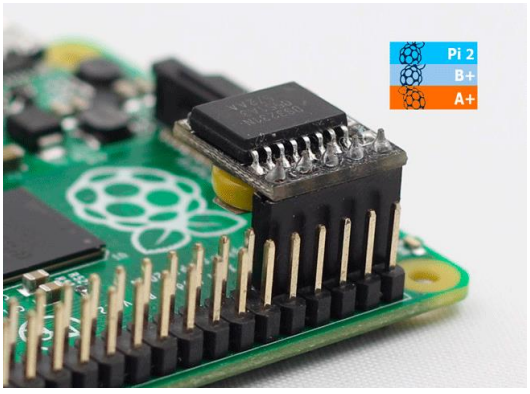


Ilustración 2.6.1

2.7. USB Dongle WI-FI

El modelo de Raspberry utilizado en el proyecto no dispone de conexión WIFI por defecto, por lo que será necesario este adaptador para dar conexión a internet al dispositivo.

Características: Velocidad de hasta 150Mps, USB 2.0

Ilustración 2.7.1

2.8. Fuente de Alimentación 220/5V 600mA

Es necesaria para alimentar la Raspberry ya que en el cuadro eléctrico no habrá un enchufe para conectarla. También existe la posibilidad de desmontar el cabezal del cargador de la Raspberry y conectar los cables de la línea eléctrica a la entrada AC del circuito interno del cargador. Después, conectar la Raspberry a su cargador normalmente.

La fuente de alimentación consiste en un simple circuito rectificador con un componente protector que aísla la parte de alta tensión de la de baja tensión. Dispone de dos conexiones a la entrada y 2 a la salida (ver ***Ilustración 2.8.2)***. En las dos conexiones de la entrada se conectan los cables de Línea y Neutro de la línea eléctrica y en los de salida, la alimentación VCC y la tierra GND de la Raspberry. (Handkraft, 2017)

Ilustración 2.8.1

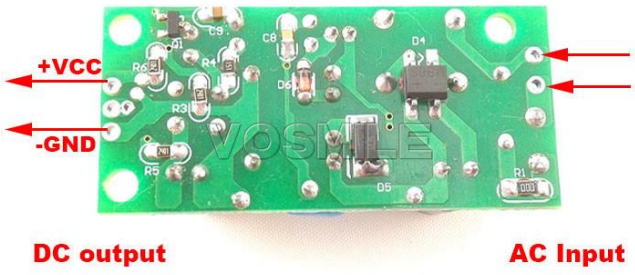


Ilustración 2.8.2

Características de la fuente:

Ondulación DC, alta eficiencia, larga vida y de la temperatura.

Protección contra cortocircuitos de salida integrado de sobrecorriente, protección contra sobretensiones (Solución de problemas de recuperación automática).

Frecuencia de voltaje de entrada: 47-63 Hz.

Voltaje de salida: DC 5V ± 0,05, salida de velocidad de regulación de voltaje: 0,2%.

600mA de corriente nominal de salida.

Rizado de salida y el ruido 75mVp-p.

Eficiencia 75-85%. diseño de bajo consumo.

Temperatura de funcionamiento 0 ℃ ~ + 70 ℃

Temperatura de almacenamiento -20 ℃ ~ + 85 ℃

Tamaño: Approx. 3 x 2 x 1.5cm.

Material: Plástico.

2.9. Cable HDMI

El cable HDMI se utiliza para transmisión de vídeo y audio en alta calidad en formato digital. En este proyecto se ha utilizado para conectar la Raspberry a una pantalla de TV con el fin de configurarla de cara a sus funciones en el proyecto.

Ilustración 2.9.1

2.10. Ratón y teclado USB

Es necesario un ratón y un teclado con conexión USB para manejar el sistema operativo de Raspberry.

Ilustración 2.10.1

2.11. Pantalla PC o TV con entrada HDMI

Es necesaria para configurar la Raspberry desde su sistema operativo. Cualquier televisor o pantalla de PC con entrada HDMI serán válidos.

3. APORTACIONES

- Para comenzar el proyecto, primero ha hecho falta comprender el funcionamiento del dispositivo Raspberry Pi sin ninguna base previa utilizando únicamente recursos online.

- El sistema operativo de Raspberry está basado en Linux, cuyo terminal tiene un lenguaje y funcionamiento propios y particulares que también han sido comprendidos y manejados mediante recursos online.

- El sistema operativo de la Raspberry ha sido instalado y configurado desde cero.

- Para manejar la Raspberry desde un PC se ha estudiado el funcionamiento de los routers WIFI, comunicación Ethernet y funcionamiento de las redes LAN.

- El mayor problema al comenzar el proyecto consistía en comunicar el dispositivo Circutor con la Raspberry. Se han investigado y probado varios dispositivos, lenguajes de programación y módulos de comunicación (adaptados a cada lenguaje) durante varias semanas antes de dar con los adecuados. Dispositivos como un conversor de RS485 a RS232 que se conecta a los pines GPIO de Raspberry han sido probados para establecer la comunicación. También se comenzó programando scripts en lenguaje #C hasta finalmente comenzar a programar en lenguaje Python, que dispone de numerosas librerías que sacan más rendimiento a las utilidades de Raspberry.

- Ante la complicación de la tarea de extracción de las medidas, se ha realizado una investigación más profunda sobre el protocolo MODBUS/RTU de Circutor. Finalmente, se ha conseguido la comunicación mediante el módulo MinimalModbus. Se han probado distintas configuraciones del módulo y del conexionado hasta dar con la configuración de MODBUS/RTU que usa Circutor.

- Después de conseguir una primera lectura de medidas, se han transcrito sus nombres y unidades a un array al mismo script de Python que extrae las medidas para atribuírselos a cada medida.

- Para el guardado de los datos se ha investigado sobre el funcionamiento, estructura y utilidades del lenguaje XML así como también sobre los módulos de Python capaces de crear y manejar archivos XML. Se ha estudiado el módulo más compatible con las necesidades del proyecto y se ha codificado en el script principal de Python para crear un archivo XML con las medidas.

- Se ha optado finalmente por la monitorización de las medidas en un smartphone o un PC. Para ello, se han estudiado todas las posibles vías de comunicación entre la Raspberry y un dispositivo remoto. Se ha estudiado y realizado pruebas con APIs como PubNub y WebRTC y servicios como NO-IP para establecer la comunicación entre ambas partes. Finalmente, se ha elegido como solución utilizar un servidor remoto como intermediario para mejorar la seguridad y no depender tanto de aplicaciones de terceros.

- Para la comunicación con el servidor se han estudiado las formas de acceder al mismo y se ha elegido un módulo de Python compatible con alguna de ellas. Se ha programado el acceso y envío de los datos al servidor utilizando el módulo FTPLIB y la documentación sobre su funcionamiento.

- Dentro del servidor ha sido creado y programado un archivo en lenguaje PHP para publicar las medidas enviadas en una web alojada en el mismo.

- Por otro lado, se ha creado una aplicación Android utilizando algunos conocimientos adquiridos en un curso sobre Android Studio y la propia documentación de programa. También se han utilizado conocimientos adquiridos de Java en la asignatura de Programación.

- Para conectar el dispositivo a la red eléctrica se ha estudiado la posibilidad de montar un circuito de alimentación para la Raspberry y finalmente se ha optado por comprar uno equivalente y compatible con la Raspberry por motivos de coste, tiempo y tamaño.

- Todos los protocolos de comunicaciones utilizados han sido estudiados y configurados para su funcionamiento y para la detección de posibles errores.

- Cada componente, así como las partes del cuadro eléctrico y las propiedades de la línea eléctrica han sido estudiados de cara a un correcto conexionado de los componentes y del dispositivo final.

- Se ha elaborado el ANEXO V donde se muestran todos los códigos programados y su explicación.

4. CONEXIONADO

Se pone de ejemplo un esquema de conexionado en una red doméstica. La parte de conexión al cuadro eléctrico de este esquema puede variar según la tipología del cuadro. El resto del conexionado será el mismo.

Esquema de conexionado:

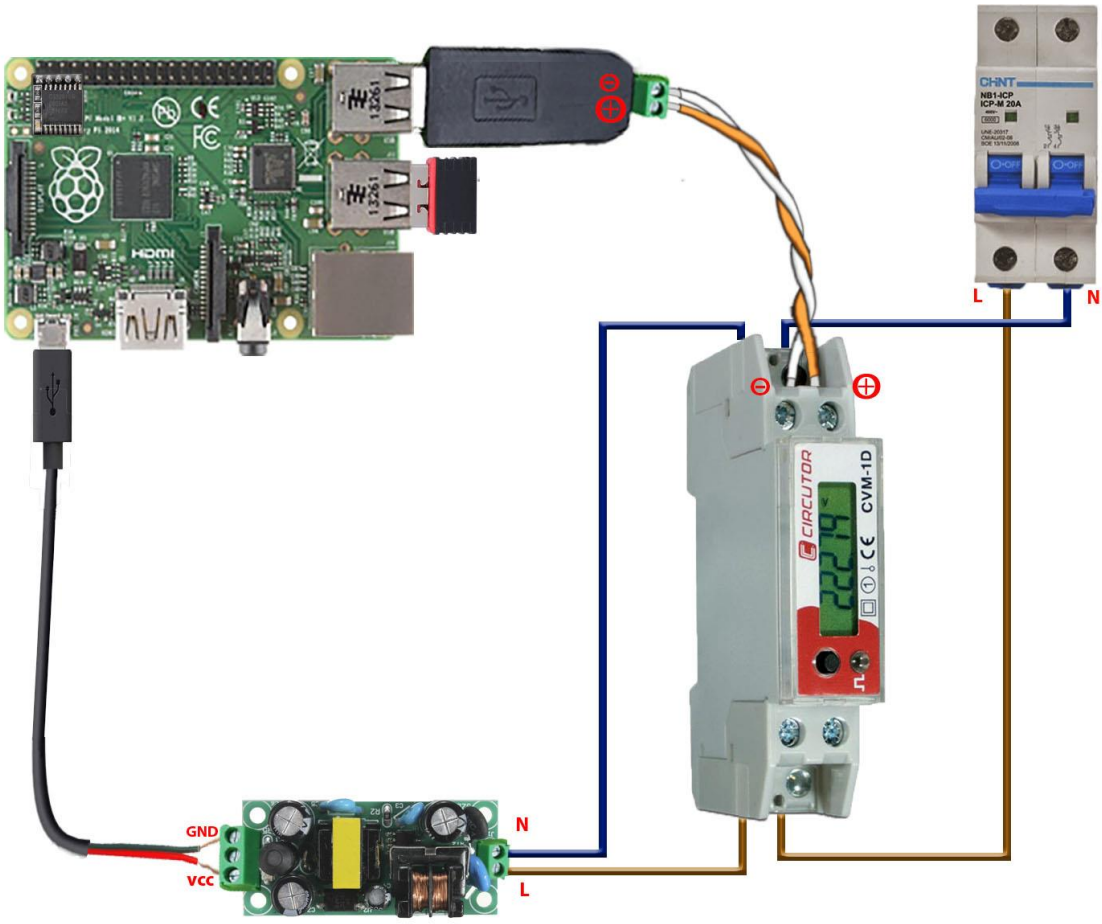


Ilustración 4.1

El interruptor que se muestra en la ***Ilustración 3.1*** es el ID. Además de alimentar el dispositivo, también alimenta las demás habitaciones (ver ***Ilustración 3.3***).

Esquema de conexionado del Circutor:

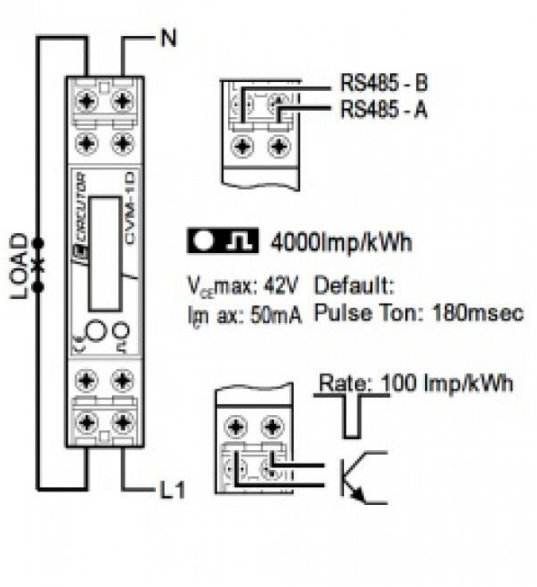


Ilustración 4.2

Acorde con la ***Ilustración 3.2***, lo primero que hay que hacer es interrumpir la línea eléctrica para intercalar el Circutor. Es decir, los cables de Línea y Neutro procedentes del interruptor ID (ver ***Ilustración 3.3***) del cuadro eléctrico irían conectados a las entradas L1 y N respectivamente, mientras que los de salida del Circutor (LOAD) irán hacia las demás habitaciones y la Raspberry.

Esquema de un cuadro eléctrico doméstico:

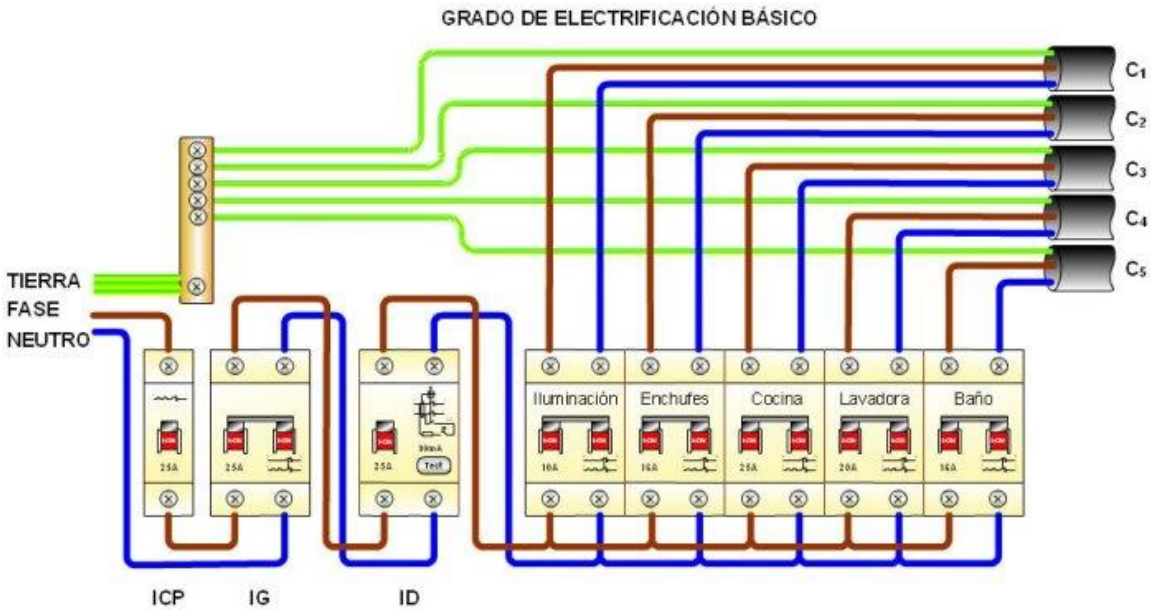


Ilustración 4.3

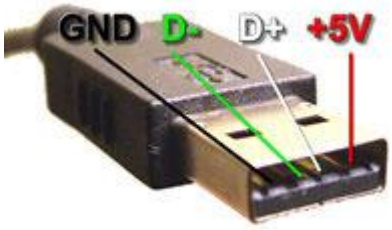


Ilustración 4.4

Ahora que la Raspberry dispone de la alimentación adecuada para su funcionamiento, se pueden conectar sus accesorios.

El Reloj RTC se conecta a los pines GPIO 1,3,5,7, el conversor USB~RS-485 y el USB Dongle WI-FI a cualquier puerto USB. Hay que tener en cuenta que el Reloj RTC debe ser conectado con la Raspberry apagada.

Hecho esto, solo faltaría conectar los cables trenzados de la comunicación RS-485. En el lado del Circutor, el cable Rojo (Datos+) se conecta a la salida A y el Blanco (Datos-) a la B (ver ***Ilustración 3.2***). En el lado del USB~RS-485, el rojo se conecta al terminal D+ y el blanco al D- (ver ***Ilustración 2.5.1***).

El conexionado acabaría así:

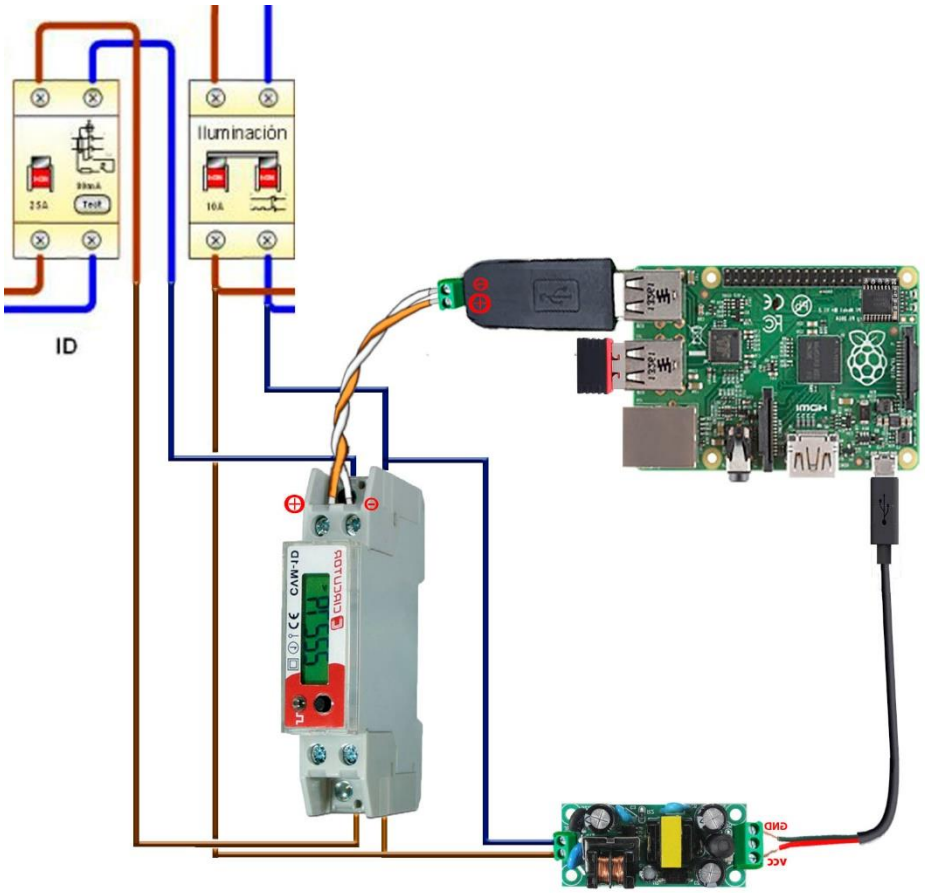


Ilustración 4.5

5. CONFIGURACIÓN DE LOS COMPONENTES

Todos los componentes deben estar configurados adecuadamente para su funcionamiento como conjunto.

4.1. Raspberry Pi

Primero, es necesario configurar la raspberry conectándola a una pantalla mediante un cable HDMI, un ratón y un teclado. Este paso es necesario para activar al menos la conexión SSH para poder manejar posteriormente la Raspberry conectándola a la red local mediante un cable ethernet.

Cuando esté encendida, lo primero que ha de hacerse es configurar la zona horaria y el teclado para establecer su idioma y poder escribir correctamente en el terminal del sistema operativo. Para ello, se accede a Menú Principal > Preferencias > Teclado y ratón.

Ilustración 5.1.1

Se selecciona Keyboard Layout y se elige el idioma.

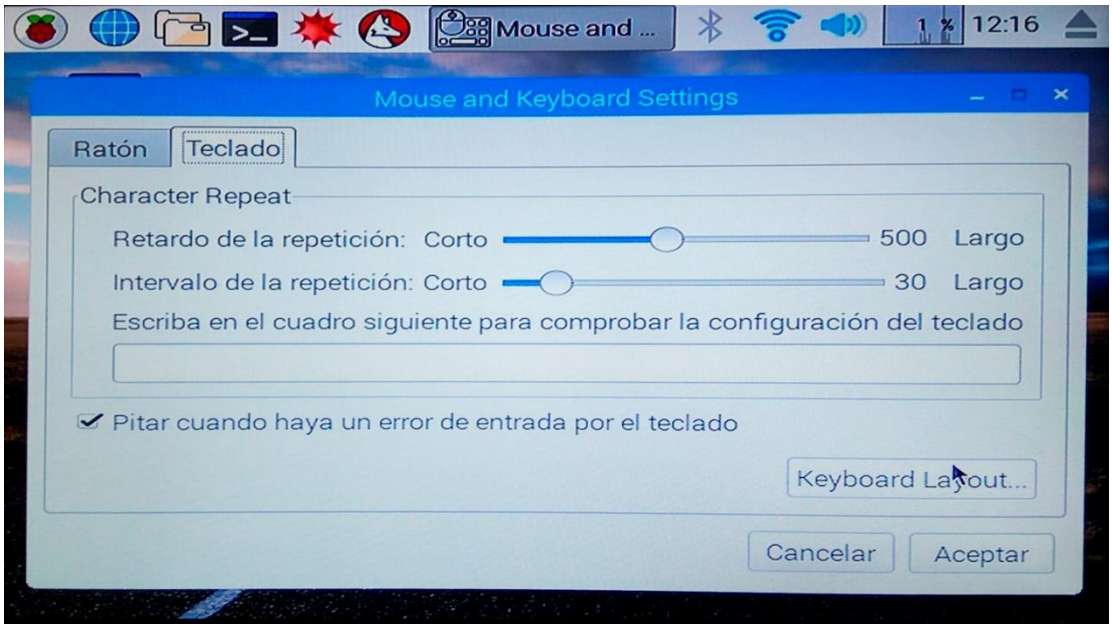


Ilustración 5.1.2

Después se accede a la configuración de la Raspberry (Menú Principal > Preferencias > Configuración de Raspberry Pi)

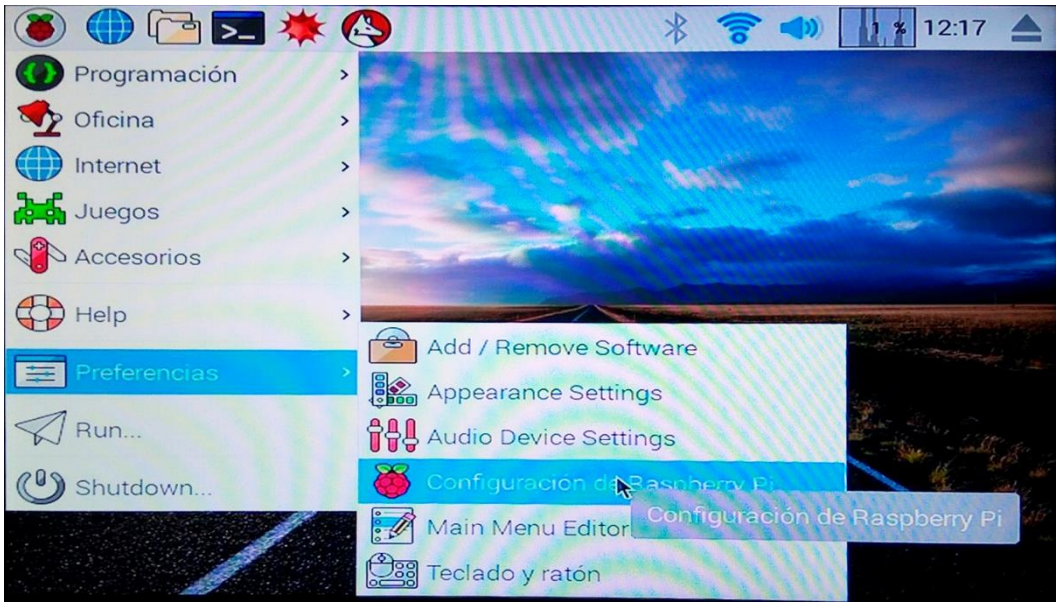


Ilustración 5.1.3

Se activan las opciones de SSH e I2C. Este paso se hace para poder comunicarse mediante el protocolo SSH con un PC para trabajar con la Raspberry sin necesidad de volver a conectarla a una pantalla, teclado y ratón. El protocolo I2C se activa para comunicarse con el reloj RTC que se conecta a los pines.

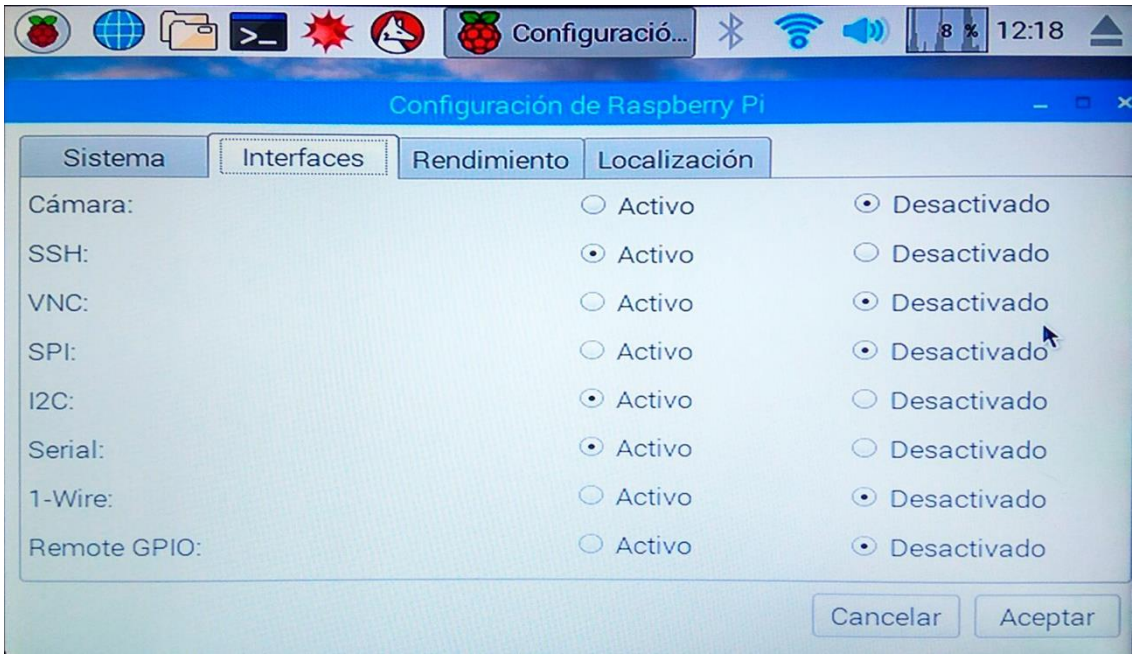


Ilustración 5.1.4

Ahora se puede conectar la Raspberry a un router mediante un cable Ethernet para manejarla desde un PC. La conexión se realiza mediante PuTTY (icono del programa ***Ilustración 4.1.6***), un software que permite operar con el terminal de la Raspberry desde Windows con el protocolo de comunicaciones SSH.

Al abrir el programa hay que introducir la IP local de la Raspberry y el puerto 22 de la comunicación SSH. La IPl se puede consultar en los clientes DHCP en las opciones del router, con un programa de escaneo de red o en la propia Raspberry ingresando el comando **ifconfig** en el terminal mediante el teclado y la pantalla conectados a ella. Se puede ver la IP en el apartado de **eth0**, **inet addr** (ver ***Ilustración 4.1.5***).

```
pi@raspberrypi ~ $ ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:b3:fc:2e
          inet addr:192.168.1.81  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4078 errors:0 dropped:0 overruns:0 frame:0
          TX packets:256 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:264593 (258.3 KiB)  TX bytes:31343 (30.6 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1104 (1.0 KiB)  TX bytes:1104 (1.0 KiB)

wlan0     Link encap:Ethernet  HWaddr 00:0f:54:12:15:97
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Ilustración 5.1.5

Ilustración 5.1.6

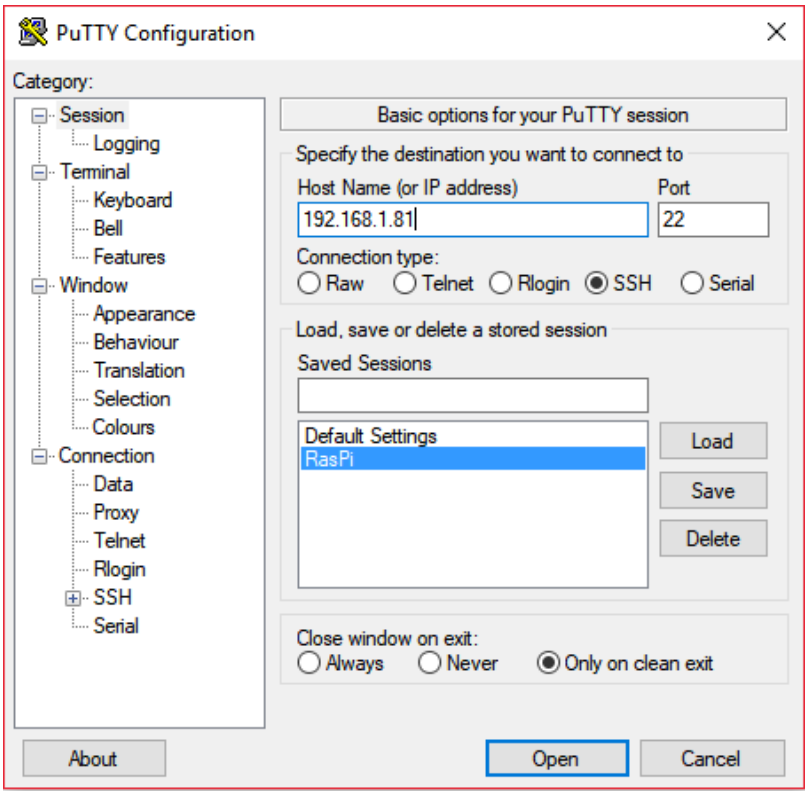


Ilustración 5.1.7

Se presiona Open y aparecerá la pantalla del terminal donde será necesario ingresar un usuario y contraseña predefinidos en la Raspberry. (Mario G., 2017)

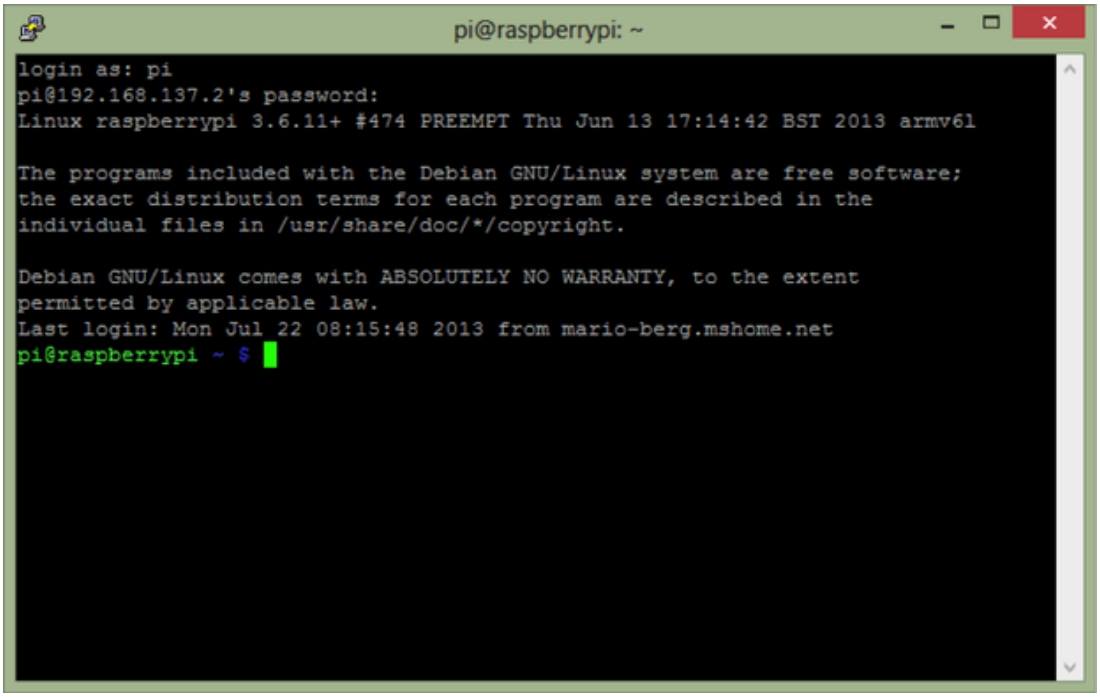


Ilustración 5.1.8

Para algunas tareas hará falta acceder al directorio ROOT, por lo que se ejecutará el siguiente comando para ejecutar posteriormente como usuario root:

```
sudo -i
```

Para el proyecto hará falta que el archivo Python que contiene el código principal se ejecute al iniciar la Raspberry. Para ello, se escriben en terminal las siguientes líneas de código.

```
crontab -e
```

Se abrirá un archivo donde se añadirá al final la siguiente línea.

```
@reboot Python /home/pi/CODIGOFINAL.py
```

4.2. USB Dongle Wi-Fi

En este proyecto se ha configurado la conexión de red para funcionar tanto por USB Wi-Fi como por Ethernet. En caso de que no se pueda hacer llegar el cable desde el cuadro eléctrico donde estará el dispositivo hasta la toma de red, se podrá conectar a la red por Wi-Fi. (ESTESO, 2017)

Se ha configurado el USB Dongle WI-FI ejecutando las siguientes líneas de código:

```
sudo nano /etc/network/interfaces
```

Se abrirá un archivo de configuración de nuestra interfaz de red donde habrá que sustituir los valores de ***mi SSID*** y ***mi password*** por los correspondientes a la red a la que se vaya a conectar:

```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0
auto wlan0

iface wlan0 inet dhcp
        wpa-ssid "mi SSID"
        wpa-psk "mi password"
```

Hecho esto, se guarda el archivo y se reinicia la Raspberry mediante el siguiente comando:

```
sudo shutdown -r now
```

Una vez reiniciada, la Raspberry ya dispone conexión a la red y se puede proceder a la instalación de los paquetes necesarios para la lectura y procesamiento de datos.

4.3. Librerías

Uno de los paquetes necesarios es ***MinimalModbus***. Se trata de una librería que sirve como herramienta para leer datos que usan el protocolo Modbus RTU. Se instala con el siguiente comando:

```
sudo pip install -U minimalmodbus
```

El módulo MinimalModbus se encarga de extraer los datos del Circutor. Para ello, primero se definen los parámetros del bus de datos y según estos, el módulo realiza la petición de un dato al Circutor este lo envía y propio módulo lo interpreta según el tipo de dato que se indique en la petición. (Python Software Foundation, MinimalModbus, 2017)

Para más detalles sobre el funcionamiento del módulo ***MinimalModbus*** dirigirse al **Anexo II.** (Berg, 2017)

Otro paquete necesario es el ***xml.minidom***. Se utiliza para construir un fichero XML con las medidas. Este paquete está incluido en las librerías estándares de Python, para actualizarlas ejecutar el siguiente comando: (Foundation Python Software, 2017)

```
sudo apt-get install python
```

Para más detalles sobre el funcionamiento del módulo ***xml.minidom*** dirigirse al **Anexo III.**

Las demás librerías utilizadas en el proyecto están incluídas en la Raspberry por defecto. Por tanto, no hará falta instalarlas.

4.4. Reloj RTC

Para configurar el Reloj RTC, primero hay que apagar la Raspberry y cuando esté apagada conectar el reloj a los pines según la ***Ilustración 2.6.1.*** (drewkeller, 2017)

Posteriormente, se ingresan en el terminal las siguientes líneas de código:

```
#Para cargar el módulo al arrancar
sudo sed -i 's/blacklist i2c-bcm2708/#blacklist i2c-bcm2708/' /etc/modprobe.d/raspi
-blacklist.conf
#Para cargar el módulo ahora
sudo modprobe i2c-bcm2708
#Para notificar a la Raspberry sobre el dispositivo
echo ds3231 0x68 | sudo tee /sys/class/i2c-adapter/i2c-1/new_device
#Para mostrar la hora del reloj RTC
sudo hwclock
```

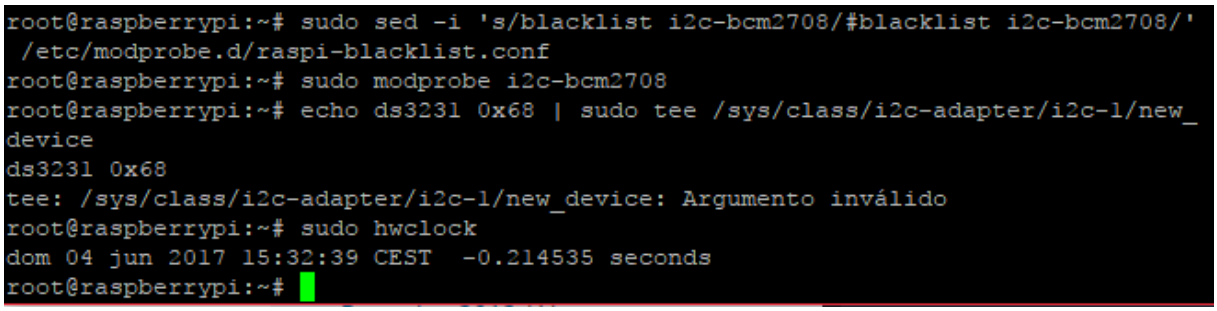


Ilustración 5.4.1

```
#Para cargar en el reloj RTC la hora desde internet
sudo ntpd -gq
sudo hwclock -w
#Para usar el reloj RTC en lugar del chip de Raspberry
sudo nano /etc/rc.local
#Se añaden al archivo las siguientes líneas
echo ds3231 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
sudo hwclock -s
#Para asegurar el paso anterior
crontab -e
#Añadir al final del archivo
@reboot sudo hwclock -s
```

Hecho esto, en el próximo inicio de la Raspberry el tiempo será tomado del reloj RTC. Esto permitirá tener siempre una hora correcta aunque no se disponga de conexión a internet o se apague la Raspberry ya que el reloj dispone de una pila.

6. FUNCIONAMIENTO GENERAL

El proceso comienza con el Circutor, que analiza la línea eléctrica y almacena los datos en sus mapas de memoria. La Raspberry recoge los datos y los almacena en un fichero XML que es enviado a un servidor remoto mediante el protocolo FTP de envío de datos utilizando la conexión WI-FI o el cable ethernet. Todo esto se realiza desde el programa principal.

El servidor, previamente configurado para interpretar el fichero XML con el fichero de **lecturas.php**, publica las medidas en una página web. Por otro lado, con la aplicación Android o con un PC, se consulta la página web y se visualizan los datos.

Esquema del funcionamiento:

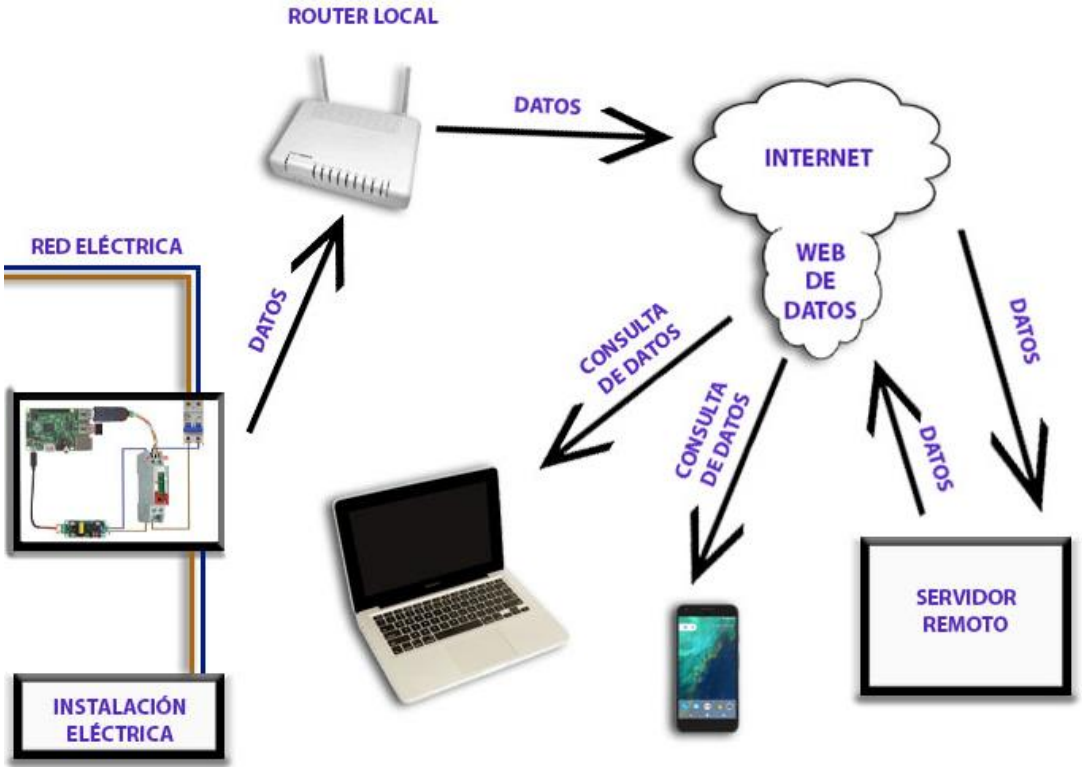


Ilustración 6.1

Se ha estudiado la posibilidad de obviar el servidor remoto y hacer una conexión P2P directamente desde la Raspberry hasta el dispositivo que vaya a leer los datos. Sin embargo, para realizar tal conexión es necesario que uno de los dos extremos actúe de servidor mientras que otro lo haga de cliente.

En caso de que la Raspberry actuase de servidor, habría que redirigir puertos del router local, lo cuál podría comprometer la seguridad de los demás dispositivos de la red además de la propia Raspberry. También podría usarse de servidor con una tarjeta sim o un módem USB, lo cuál añadiría coste al proyecto.

En caso de usar el dispositivo móvil o PC como servidor, además de tener los mismos problemas de seguridad de redirección de puertos, la dirección del dispositivo iría cambiando (debido a que se no siempre está conectado a la misma red) y sería ilocalizable por la Raspberry.

También existía la posibilidad de utilizar un servidor de terceros o de prueba encontrados en internet. Algunos de ellos eran de pago mientras que otros simplemente no son compatibles con ningún módulo de Raspberry.

Se ha optado finalmente por utilizar un servidor disponible para tareas sencillas que está configurado para alojamiento web y que se encuentra en una red con medidas de seguridad más elaboradas que las de una red particular. El servidor tiene la particularidad de que sólo permite el acceso a un fichero público sin datos importantes para beneficio de la seguridad.

6.1. Adquisición de datos

Para recoger los datos almacenados en el Circutor, la Raspberry debe enviar la solicitud de un dato almacenado en un lugar de la memoria del Circutor (**ANEXO I**) a través del bus de datos RS-485. El Circutor enviará mediante el protocolo MODBUS/RTU el dato almacenado en el mapa de memoria escogido.

Todo el procedimiento lo realiza el módulo MinimalModbus, que se programa en el programa principal. Reconoce el protocolo MODBUS/RTU, realiza la petición e interpreta el bus de datos recibido.

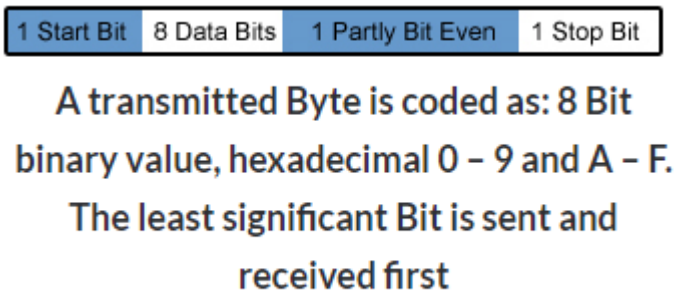


Ilustración 6.1.1

En la ***Ilustración 5.1.1*** se muestra la disposición de los datos del bus RTU que viaja a través del RS-485. Antes de la información se recibe un bit de comienzo, después 8 bits de datos comenzando por el menos significativo, seguidamente un bit de paridad que indica si el numero de bits a 1 en un conjunto es par o impar. Finalmente se transmite un bit de final de bus.

Esta forma de transmisión de datos es interpretada por el módulo MinimalModbus, que recoge los bits de información y los transforma en número decimal.

A la hora de recoger los datos hay que tener en cuenta el formato del dato, si tiene signo o no, su dirección en la memoria y el parámetro function code (ver **Anexo II**). Los datos se toman en formato Long con precisión de 32 bits para transformarlos a Float con el objetivo de representar los decimales de cada medida.

Para ver las líneas de código para configurar la adquisición de datos ver **ANEXO V,** apartado 1.

Del Circutor se extraerán todas las medidas disponibles y se guardrán en un array de datos.

6.2. Guardado de los datos

Para guardar los datos se ha elegido el formato XML debido a que este tipo de ficheros son usados con frecuencia en manejo de datos en páginas web y bases de datos. (EXES, 2017)

El lenguaje XML es un lenguaje simplificado que busca utilizar la información de una forma estructurada y adaptada a internet. Los documentos XML constan de varias partes:

Prólogo

Los documentos XML suelen comenzar por unas líneas en las que se declara el archivo como XML, el tipo de documento y algún comentario o instrucción de procesamiento. El prólogo se puede omitir. Ejemplo de prólogo:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Cuerpo

El cuerpo no es opcional en un documento XML. Se compone de elementos y debe estar formado por un elemento raíz. Cada elemento puede tener atributos contener varios elementos dentro.

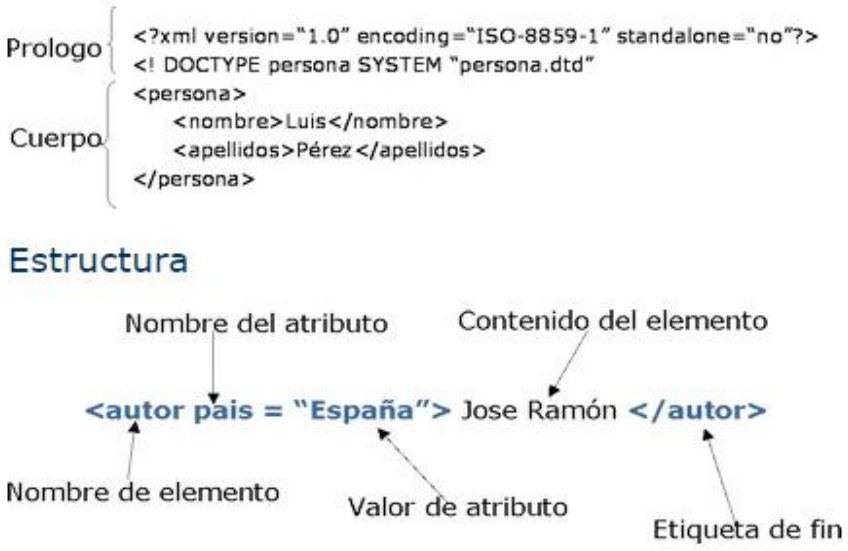


Ilustración 6.2.1

Los datos se escriben en uno de los subelementos del fichero XML. Se crea primero el elemento root (documento del cliente) cuyo atributo es el DNI del cliente. Posteriormente se crea un subelemento en root de fecha con sus correspondientes atributos de año, día y mes. Dentro del subelemento fecha se crea el de medidas, cuyos atributos son las medidas y sus respectivos nombre, valor y unidad.

La fecha de las medidas se toma desde el Reloj RTC mediante el protocolo I2C, que consiste en un bus de datos entre la Raspberry y el circuito conectado a sus pines.

El fichero se guardará con el nombre de la fecha en la que se toman las medidas y se almacenará en la carpeta de trabajo de la Raspberry (/home/pi/) hasta que se sobreescriba por el fichero de la siguiente toma de medidas.

6.3. Envío de datos

Una vez creado el fichero XML, se envía al servidor remoto usando el protocolo FTP. También se añade el fichero de **EVENTOS.txt** al envío para la gestión de errores.

La librería utilizada para el envío de datos es **ftplib**, que da la posibilidad de enviar archivos en formato binario a una dirección IP remota utilizando el protocolo FTP. Para ver esta y más funcionalidades dirigirse al **ANEXO IV**.

El protocolo permite comunicarse con una dirección IP fija en internet. En este caso se tratará de un servidor con una IP fija que requerirá un usuario y una contraseña para que sea accesible. Los tres datos estarán especificados en el código principal de la Raspberry y gestionados por el módulo **ftplib**.

La Raspberry envía el archivo de medidas antes de sobreescribirlo por uno nuevo. En el servidor se van almacenando todas las medidas y un único fichero de eventos que se va sobreescribiendo y al que se van añadiendo nuevas líneas.

Evidentemente, para que el envío se produzca es necesario que la Raspberry disponga de conexión a internet y que el servidor sea accesible.

6.4. Visualización de los datos

La visualización de los datos se puede realizar desde la aplicación Android o desde la web alojada en el servidor de destino de los datos.

Ambas formas requieren superar un proceso de autenticación de usuario para recibir las medidas correctas.

La aplicación Android se ha hecho mediante el software Android Studio, cuyos principales lenguajes de programación son Java y XML.

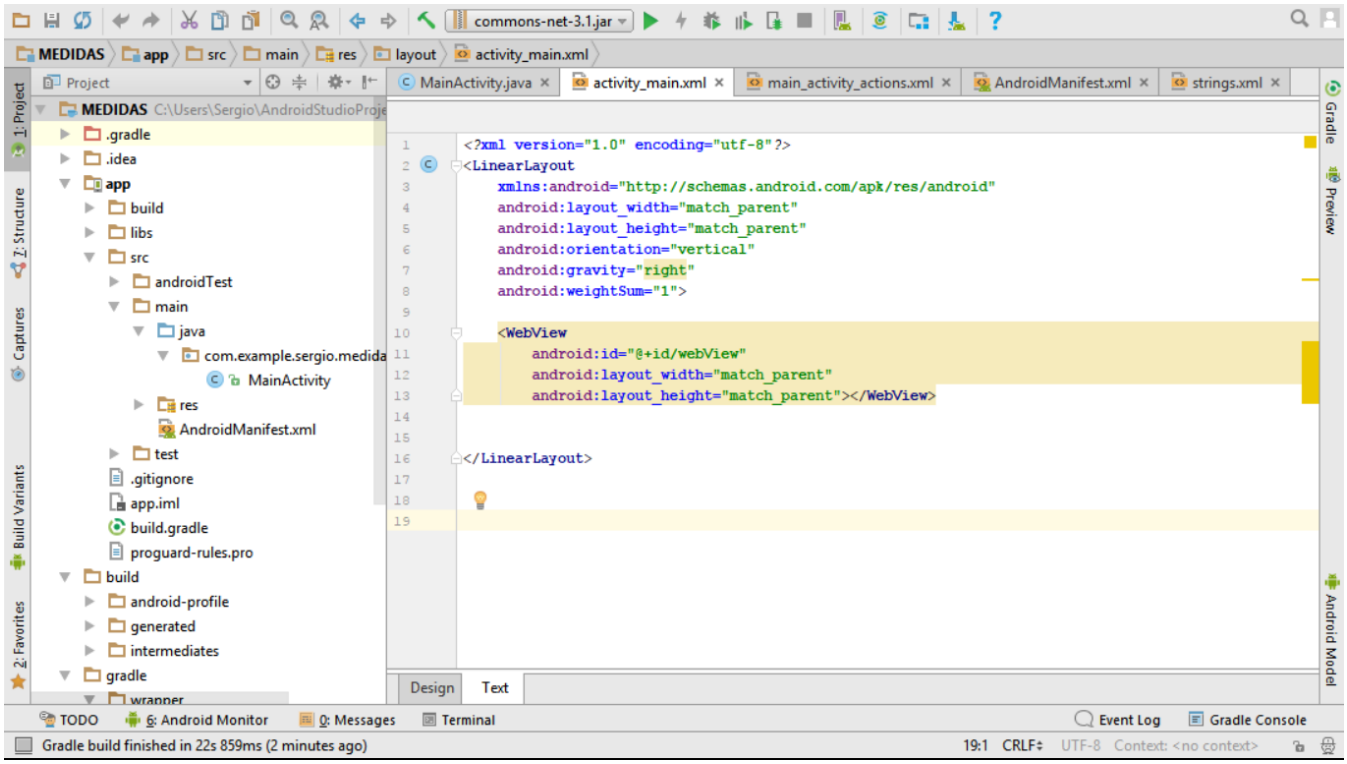


Ilustración 6.4.1

En la ***Ilustración 5.4.1*** se muestra la pantalla principal de Android Studio con los archivos necesarios para la aplicación.

Android Studio utiliza varios archivos para representar lo que se desea mostrar en pantalla. El archivo principal es el **MainActivity.java**, donde se programarán las actividades principales y las acciones que se realizan en cada actividad.

Existe una carpeta llamada **layout**, donde se encuentran los diseños de pantallas, botones, cuadros de texto, etc. En esta carpeta se encuentra el archivo **activity_main.xml**, donde se establece el diseño de la página principal en lenguaje XML.

La aplicación consistirá en la visión web de la página creada para mostrar las medidas y un botón de recargar para actualizarla.

Ver los códigos de la aplicación en **ANEXO V**, apartado 5.

Una vez completada la aplicación se ha guardado como archivo APK y se ha transferido a un dispositivo Android.

La aplicación tiene un aspecto como este:

Ilustración 6.4.8

Al abrir la aplicación se mostrará el botón de refrescar en la barra de acción y la página web que contiene la pantalla de verificación de usuario:

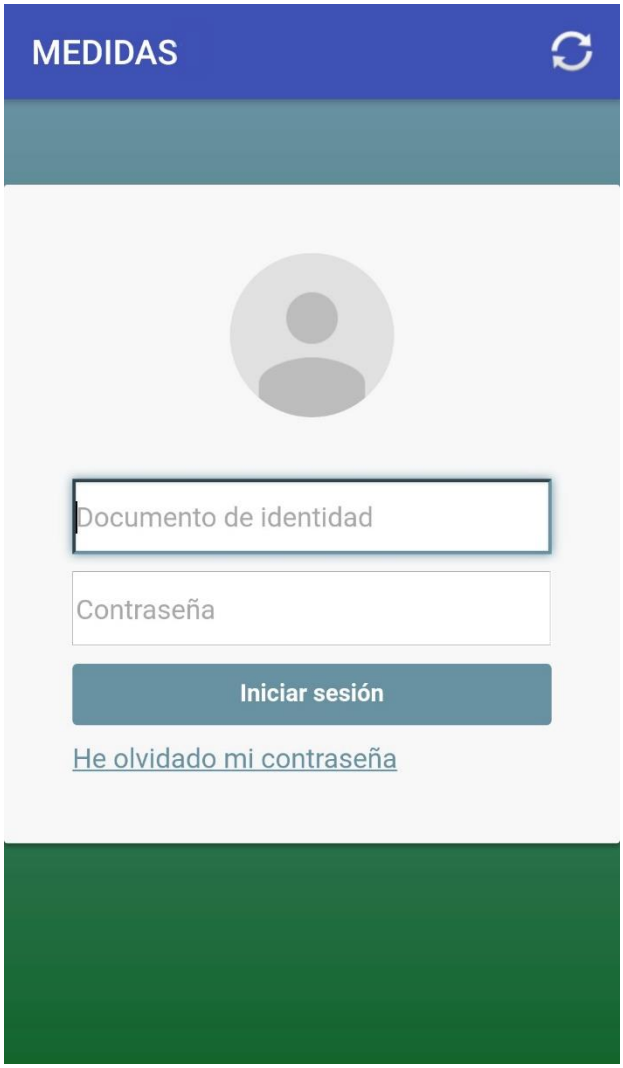


Ilustración 6.4.9

Al ingresar los datos en la pantalla de login aparecerán las medidas enviadas desde la Raspberry.

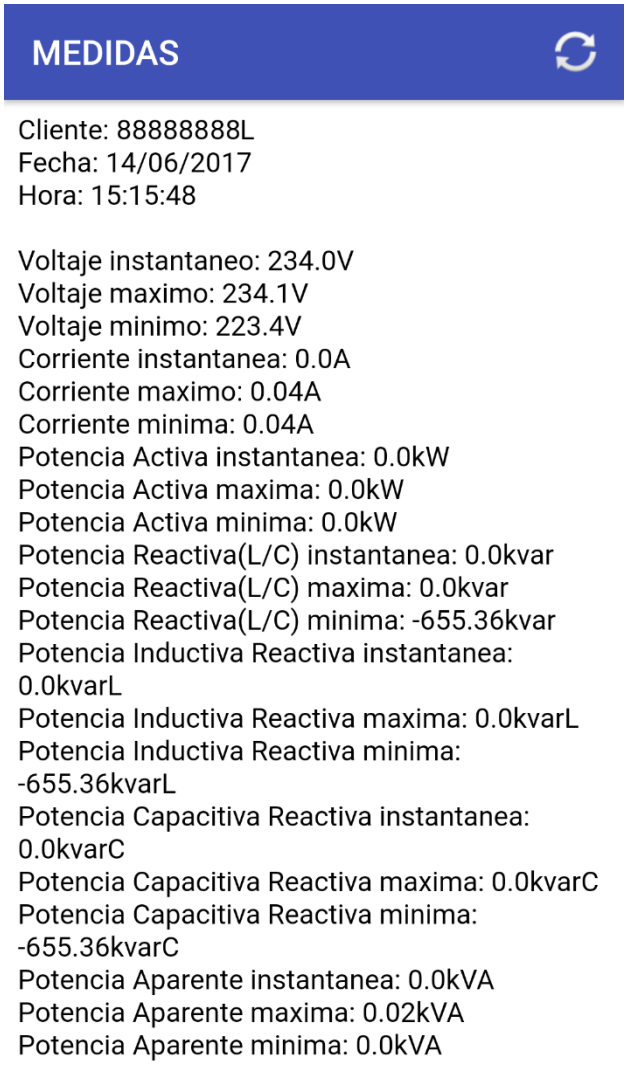


Ilustración 6.4.10

Las medidas se podrán actualizar al valor más reciente pulsando el botón de refrescar.

La aplicación es compatible con cualquier teléfono o tablet Android con versión superior a 4.0.3, lo cual la hace compatible con el 90% de los dispositivos del mercado.

En el servidor remoto se ha creado un fichero de **lecturas.php** (ver **ANEXO V**, apartado 6) para leer el último fichero de medidas enviado al servidor y disponerlo en la página web de una forma más legible que el formato XML en el que se envía.

En el servidor remoto también hay un fichero que maneja todas las medidas almacenadas para montar gráficos y tablas de consumo.

Visualización de las medidas en PC ingresando directamente la página en el buscador.

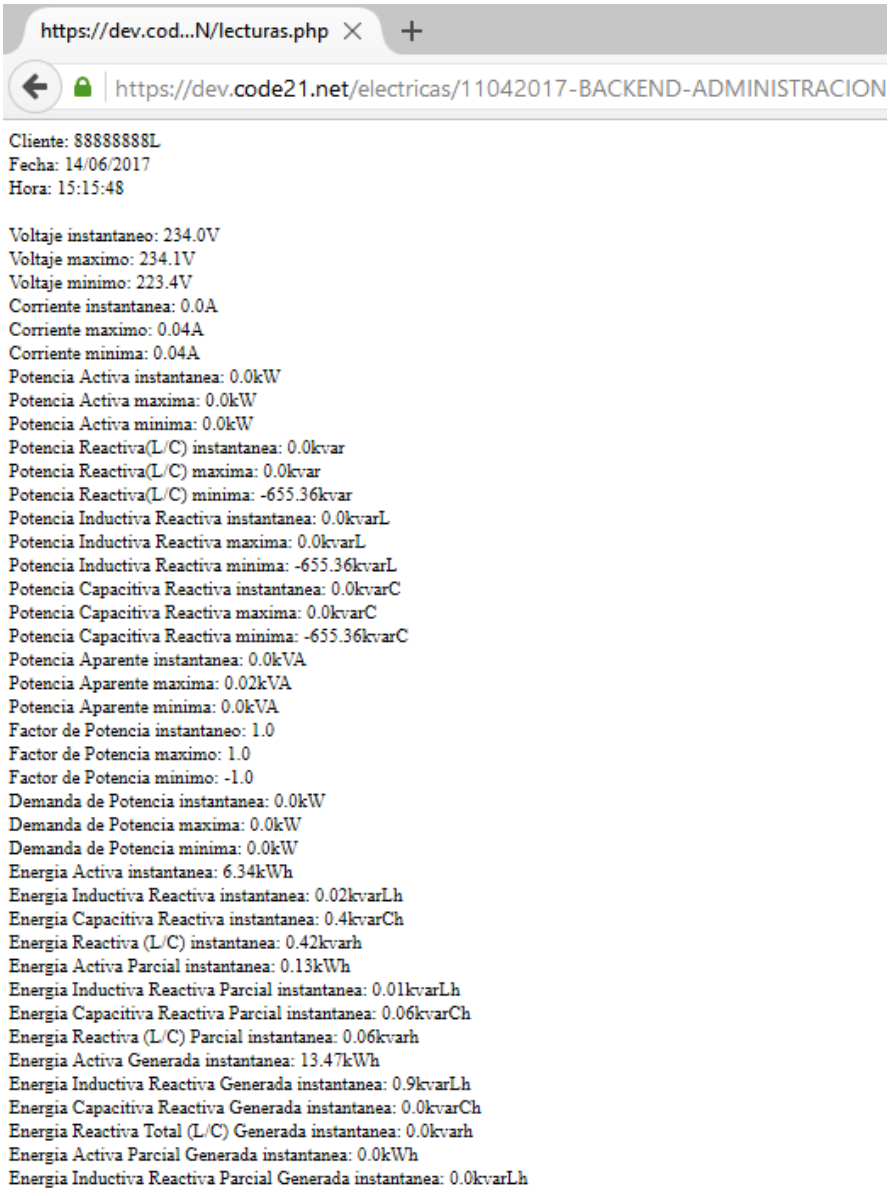


Ilustración 6.4.11

6.5. Gestión de errores

Es necesario detectar los posibles errores ya que, en caso de que se produzca alguno, la Raspberry detendrá la ejecución del código principal y no volverá a reanudarla hasta que se reinicie.

Estableciendo la detección de errores, el código se pausará cuando se produzca el error y se reanudará cuando este se arregle.

Todos los eventos de errores y reanudación del proceso se podrán observar en el archivo **EVENTOS.txt** almacenado en cada ciclo de medición en el servido remoto.

6.2.1 USB desconectado

Puede ocurrir que al manejar la Raspberry sin mucho cuidado, se extraiga el USB donde se conectan los cables del bus RS-485.

Para detectar este error se han añadido unas líneas de código que detendrán el proceso cuando no se detecte el USB del proyecto.

Esto se hace extrayendo el número ID del dispositivo USB para compararlo con todos los dispositivos conectados a la Raspberry. Si no se encuentra un dispositivo con el mismo número ID, significará que no está conectado y no se seguirá con el proceso de extracción de los datos.

Al producirse este error, Se notificará en el archivo **EVENTOS.txt** junto con su fecha y hora. Cuando se vuelva a conectar, se notificará el reestablecimiento de la conexión y se reanudará el proceso.

Ver código en **ANEXO V,** apartado 2.

6.5.2 Circutor inaccesible

Este método detecta cuando el programa envía una petición de datos al Circutor y no obtiene respuesta u obtiene una respuesta que no corresponde con el formato de dato solicitado.

Este procedimiento detecta dos tipos de errores: El Circutor ha sido desconectado de su alimentación o los cables que lo unen al USB han sido desconectados del USB o del Circutor.

Cuando se detecta el error, el proceso se detiene y se comunica el fallo con la fecha y hora en el archivo de **EVENTOS.txt**. El proceso se reanuda cuando vuelve a haber recepción correcta de datos y se comunica también en el archivo de eventos.

Ver código en **ANEXO V,** apartado 3.

6.5.3 Servidor inaccesible

A veces la conexión internet puede ser inestable y sufrir algún corte. Esto puede suceder tanto en el lado de la Raspberry como en el lado del servidor.

También puede ocurrir que el servidor esté ocupado o que no se esté mostrando en la red por algún tipo de fallo interno.

Es necesario detectar el error del servidor inaccesible para que no se produzca el fallo en el programa principal de Raspberry, que detendrá el proceso y no volverá a ejecutar el código hasta que se reinicie la Raspberry.

Cuando se produzca el fallo, el programa lo notificará en el fichero de **EVENTOS.txt** junto con la fecha y hora del evento y pausará la ejecución del código.

Ver código en **ANEXO V,** apartado 4.

7. PRESUPUESTO APROXIMADO

Los componentes utilizados en el proyecto cumplen al 100% con los objetivos del mismo pero no son los más óptimos ni los más económicos, simplemente se han utilizado porque eran los que había disponibles. Por tanto, el presupuesto del proyecto se puede reducir considerablemente.

Todos los precios recogidos son el precio de venta a particulares.

Precios de los componentes:

Raspberry Pi……35€

Circutor CVM-1D……107€

Dongle WI-FI USB……...3€

Reloj RTC……..2€

RS-485 to USB……2€

Fuente AC/DC 220V/5V……..2€

Cables de conexionado……2€

Es el precio de la Raspberry incluye la tarjeta SD necesaria para instalar el sistema operativo. Puesto que en este proyecto no se utilizará cargador, valdrá con la Raspberry y algún cable adicional incluido en cables de conexionado.

El Circutor sería sustituible por otros componentes de misma funcionalidad pero de precio en torno a los 15€.

PRESUPUESTO TOTAL APROXIMADO……153€

8. MEJORAS

La principal mejora que se podría aplicar al proyecto es diseñar una placa base desde cero adaptada a las necesidades del propio proyecto. Se integraría en una misma placa el dispositivo medidor y el procesador de datos. Así aumentaría la seguridad y robustez del dispositivo a la par que reduciría los costes del proyecto. También sería una gran ventaja de cara a la comercialización del dispositivo ya que minimizaría la dependencia de productos de terceros como Raspberry o Circutor a componentes más básicos y fiables de cara a un funcionamiento prolongado.

Otra de las mejoras podría ser cambiar el modelo de Raspberry por una más avanzada, con wifi incorporado y compatibilidad con librerías más avanzadas. Este cambio también daría la posibilidad de hacer capturas de pantalla ya que en la Raspberry utilizada no se permite debido a una incompatibilidad.

También sería conveniente sustituir el Circutor por un dispositivo más económico o incluso montar un circuito para transformar los parámetros de la línea eléctrica a valores legibles por los pines GPIO de la Raspberry.

Si reducimos la funcionalidad del dispositivo a la red local (no se podría acceder a las medidas desde otra red), otra posibilidad del proyecto sería crear un servidor Apache en la Raspberry que sólo sería accesible desde la red local a la que esté conectada la Raspberry. Esto aumentaría la seguridad del dispositivo al no ser visible más allá del router local y permitiría acceder a las medidas de la misma manera que con un servidor externo.

Otra posibilidad sería extraer los datos por Bluetooth, sacrificando también la funcionalidad de disponer de las medidas desde cualquier lugar con internet, en beneficio de la seguridad. El adaptador Bluetooth tiene un precio similar al Wi-Fi.

El protocolo utilizado para la transmisión de datos hacia el servidor externo es el FTP, el cual es sencillo y rápido pero también inseguro puesto que no dispone de ningún tipo de cifrado de datos. Sería una buena mejora añadir una encriptación TLS/SSL, que protege el usuario y la contraseña y opcionalmente los datos. Esta mejora no está disponible en el servidor utilizado.

También sería buena opción sustituir el protocolo FTP por uno más seguro como el SCP o SFTP (no disponible en el servidor externo utilizado en el proyecto), que permiten enviar los datos pero cifrando todo el tráfico.

Todas las conexiones de petición de datos pasan por el servidor remoto. Sin embargo, se podría añadir un código para que cuando la Raspberry y el dispositivo que consulta los datos estén en la misma red se realice la conexión de forma local y se extraigan los datos sin hacerlos pasar por el servidor externo. Esto sería conveniente en caso de que queramos reducir el tránsito en el servidor. En el caso de este proyecto no será necesario.

9. CONCLUSIONES

Se ha montado un dispositivo que analiza la red eléctrica a partir de varios aparatos que han sido configurados para funcionar como conjunto.

El dispositivo se colocará en el cuadro eléctrico del edificio y se conectará a internet por Wi-Fi o cable Ethernet utilizando cualquier red del edificio.

Uno de los objetivos principales del trabajo era observar el consumo (Potencia Instantánea) de la red eléctrica. Este dato es uno de los recogidos por el dispositivo y se puede observar su evolución mediante cualquiera de los métodos de visualización anteriormente citados.

Para realizar el trabajo era necesario utilizar un microcontrolador o un computador que maneje los datos, en este caso se ha utilizado la Raspberry. La Raspberry extrae, guarda y envía los datos y gestiona los errores. Los lenguajes con los que se ha programado la Raspberry son Linux para el terminal, Python para el código principal y XML para el guardado de datos.

Se ha creado una plataforma o página web en el servidor al que envía los datos la Raspberry con el objetivo de visualizarlos online. Este paso se ha realizado con un archivo programado en lenguaje PHP que interpreta los ficheros XML enviados y los publica en una página web.

Adicionalmente, se ha creado una aplicación Android para aportar versatilidad a la monitorización de los datos. En este paso se han utilizado los lenguajes Java y XML.

Una vez conectado el dispositivo se podrá visualizar el consumo energético de la instalación desde cualquier parte del mundo con acceso a internet.

ANEXO I. CIRCUTOR

The CVM-1D is an instrument that measures, calculates and displays the main electrical parameters in industrial and domestic single-phase power networks. The measurement is an RMS value made by direct measurement of the current and voltage. The measured and calculated parameters are shown in the table of variables.

This document is the user and operating manual for the **CVM-1D** device. You can download the manual from the CIRCUTOR web site if necessary:

www.circutor.com

You must disconnect the unit from the power supply before performing any maintenance operations, connection modifications, repairs, etc. When a failure in the unit's operation or in its protection is suspected, the unit must be removed from service. The design of the unit makes it easy to replace in the event of a fault.

1.- Button

The CVM-1D analyzer's front panel has a six digit LCD display as well as a button function that enables the user to navigate through the different display screens of the primary electrical variables.

This button is used for two types of navigation, depending on how it is pressed:

SHORT PRESS: This type of press occurs when the user maintains the function button pressed for less than two seconds. With a short press, the device moves through the different navigation screens, displaying all the electrical parameters on the display (refer to section 2.- Display). In numerical setting, the short press allows a cyclical increase the value of the digit.

LONG PRESS: This type of press occurs when the user maintains the function button pressed for longer than two seconds. With a long press, the device intermittently displays the maximum and minimum values of the variable that is displayed at that moment. With a long press on the partial power values, the unit resets these values. In numerical settings, long press allows the left lateral selection and posterior digit validation. If the selected value is not correct, the digits will blink indicating the user to enter a correct value (refer to section 5.- Setup, allowed values).

2.- Display

The unit's front panel incorporates an LCD display with six digits. By repeatedly pressing the function button located on the front panel, the unit displays the different measured electrical parameters and the symbol corresponding to the displayed variable.

3.- Measurement

The Power analyzer CVM-1D is a measuring unit with four quadrants, which is valid for conventional power consumption systems and systems where a certain type of generation source exists.

For this, the unit is able to display the primary electrical variables with their sign (kW and kVA), displaying the direction of the current.

3.1.- Electrical Variables

The electrical variables are displayed on the unit by a rotating display screen system. This enables the user to quickly display all the electrical variables by repeatedly pressing the function button.

Upon start-up and after the unit is connected to an auxiliary power supply, the unit displays the firmware versionfollowed by the following electrical variables:

3.1.1.- Phase – neutral voltage

Voltage between the phase and the neutral with a maximum resolution of 1 decimal points (235.1 V). By pressing and holding the voltage value, the unit displays the maximum value by fast flashing and the minimum recorded value by slow flashing.

3.1.2.- Current

Briefly pressing the function button, the unit displays the Current with a maximum resolution of 2 decimal points (15.24 A).

By pressing and holding the voltage value, the unit displays the maximum value flashing quickly and the minimum recorded value flashing slowly.

3.1.3.- Active Power

By briefly pressing the function button, the unit displays the Active Power with a maximum resolution of 2 decimal points (3.24 kW). If the measurement is taken at the output of a power generator load, the parameter is displayed with a negative sign.

By pressing and holding the active power value, The unit displays the maximum value flashing quickly and the minimum recorded value flashing slowly.

3.1.4.- Reactive Power

By briefly pressing the function button, the unit displays the Reactive Power with a maximum resolution of 2 decimal points (2.12 kvar).

The unit displays the work quadrant with its sign; if the value is positive, it displays the Inductive Reactive Power (kvarL); if the value is negative, it displays the Capacitive Reactive Power (kvarC).

By pressing and holding the reactive power value, The unit displays the maximum value flashing quickly and the minimum recorded value flashing slowly.

3.1.5.- Apparent Power

By briefly pressing the function button, the unit displays the Apparent Power with a maximum resolution of 2 decimal points (5.10 kVA). If the measurement is taken at the output of a power generator load, the parameter is displayed with a negative sign.

By pressing and holding the apparent power value, the unit displays the maximum value flashing quickly and the minimum recorded value flashing slowly.

3.1.6.- Maximum demand

By briefly pressing of the function button, the unit displays the Maximum Demand. The maximum demand is calculated using the sliding window method for a time set by the user via the configuration setup.

The maximum demand can be calculated with respect to two selectable variables (A - kW). The unit is set as follows by default:

a) md code: Active Power (KW)

b) Period: 15 minutes

By pressing and holding the maximum demand value, the unit displays the maximum value flashing quickly and the minimum recorded value flashing slowly.

3.1.7.- Power Factor

By briefly pressing the function button, the unit displays the Power Factor with a maximum resolution of 2 decimal points (-0.99). The unit displays the work quadrant with its sign (see diagram Sign convention).

By pressing and holding the power factor value, the unit displays the maximum value flashing quickly and the minimum recorded value flashing slowly.

3.1.8.- Active Energy

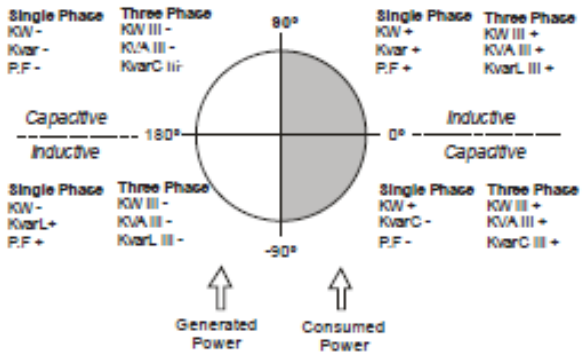
By briefly pressing of the function button, the unit displays cons before the Active Energy Consumption with a maximum resolution of 1 decimal point, with a background scale of 99999.9 kWh.

3.1.9.- Reactive Energy

By briefly pressing the function button, the unit displays the Reactive Energy Consumption with a maximum resolution of 1 decimal point, with a background scale of 99999.9 Kvarh. The unit displays the work quadrant with its sign (see diagram Sign convention).

3.1.10.- Partial Active Energy
By briefly pressing of the function button, the unit displays par before the Partial Active Energy Consumption with a maximum resolution of 1 decimal point and a background scale of 99999.9 kWh. By pressing and holding the partial active energy value, the unit resets both partial meters (partial active energy consumption and partial reactive energy consumption).
3.1.11.- Partial Reactive Energy
By briefly pressing of the function button, the unit displays the Partial Reactive Energy with a maximum resolution of 1 decimal point, with a background scale of 99999.9 kVAr·h. The unit displays the work quadrant with its sign (see diagram Sign convention). By pressing and holding the partial active energy value, the unit resets both partial meters (partial active energy consumption and partial reactive energy consumption).
3.1.12.- Generated Active and Reactive Energy
By activating the measurement in four quadrants via the setup, the analyzer displays gen before the Generated Active and Reactive Energy as well as a second block of partial meters.
By pressing and holding the partial meters display, the unit resets both partial meters (generated partial active energy and generated partial reactive energy).

4.- Sign convention



5.- Setup
To enter the configuration setup, display an energy variable (any), and make a long press to the function button, to display on the screen SETUP.
By pressing and holding, the unit displays the different configuration sections, and by pressing briefly, you can change their values.
a) nPEr: peripheral number 001...254 - Default (1)*
b) bAud: rate 2400-4800-9600-19200 - Default (19.200)*
c) quad: 2 quadrants / 4 quadrants
d) maximum demand configuration

- MD VAR: 3 (kW- active power) / 2 (A - current)
- MD per: 1...60 minutes
e) F.OUT: PULSE (pulse function) / ALARM (alarm function)*

PULS – FUNCtion energy impulse:
- P VAR: 10, 11, 12, 13 (consumed)) 18., 19, 20 ,21 (generated)
- P TIME: 40...200 ms. (pulse duration)
ALARM – alarm function:
- A VAR: 1...9 (instant variable)
- A MAX: maximum value
- A MIN: minimum value
- ADLAY: delay connection and disconnection (0...60 sec)

In the alarm function, the digital output is maintained open between the maximum and minimum value. In the case of programming an inverse logic (normally closed), invert the maximum and minimum values in the Setup menu.
The p VAR and a var CODES are specified in the Modbus/RTU memory Map table, in the Var column. If you do not wish to programme a variable, select 00.
To validate the modified data in setup, ensure you can view all of the display screens by pressing and holding, until completing all of the configuration options. At the end of the process, the unit validates and saves the changes that have been carried out.
If the configuration process is not completely finished and after not having pressed the function key for 10 seconds, the unit returns to the display screen, and exits the setup menu without saving the data that has been modified by the user.
*Options a) and b), are listed in model RS485, since it expressly references the device's communication parameters. Option e) is fixed to “pulse-active energy consumed” in the MID certified device.The rest of the options are listed in all the CVM-1D range references.

6.- Modbus/RTU memory map

Parameters	Symbol	Var	Instantaneous	Maximum	Minimum	Units
Voltage	V	1	0000-0001	0032-0033	0044-0045	V x10
Current	A	2	0002-0003	0034-0035	0046-0047	A x100
Active Power	kW	3	0004-0005	0036-0037	0048-0049	± kW x100
Reactive Power (L/C)	kvar	4	0006-0007	0038-0039	004A-004B	± kvar x100
Inductive Reactive Power	kvarL	5	0008-0009	003A-003B	004C-004D	± kvarL x100
Capacitive Inductive Power	kvarC	6	000A-000B	003C-003D	004E-004F	± kvarC x100
Apparent power	kVA	7	000C-000D	003E-003F	0050-0051	± kVA x100
Power Factor	PF	8	000E-000F	0040-0041	0052-0053	PFx100
Maximum Demand	kW / A	9	0010-0011	0042-0043	0054-0055	kW / A x100
Active energy	kW·h	10	0012-0013	-	-	kW·h x100
Inductive Reactive Energy	kvarL·h	11	0014-0015	-	-	kvarL·h x100
Capacitive Reactive Energy	kvarC·h	12	0016-0017	-	-	kvarC·h x100
Reactive Energy (L/C)	kvar·h	13	0018-0019	-	-	kvar·h x100

Parameters	Var	Symbol	Instantaneous	Maximum	Minimum	Units
Partial Active Energy	14	kW·h	001A-001B	-	-	kW·h x100
Partial Inductive Reactive Energy	15	kvarL·h	001C-001D	-	-	kvarL·h x100
Partial Capacitive Reactive Energy	16	kvarC·h	001E-001F	-	-	kvarC·h x100
Partial Reactive Energy (L/C)	17	kvar·h	0020-0021	-	-	kvar··h x100
FOUR QUADRANTS MEASUREMENT						
Generated Active Energy	18	kW·h	0022-0023	-	-	kW·h x100
Generated Inductive Reactive Energy	19	kvarL·h	0024-0025	-	-	kvarL·h x100
Generated Capacitive Reactive Energy	20	kvarC·h	0026-0027	-	-	kvarC·h x100
Generated Total Reactive Energy (L/C)	21	kvar·h	0028-0029	-	-	kvar·h x100
Partial Generated Active Energy	22	kW·h	002A-002B	-	-	kW·h x100
Partial Generated Inductive Reactive Energy	23	kvarL·h	002C-002D	-	-	kvarL·h x100
Generated Capacitive Reactive Energy	24	kvarC·h	002E-002F	-	-	kvarC·h x100
Partial Generated Total Reactive Energy (L/C)	25	kvar·h	0030-0031	-	-	kvar·h x100

7.- CVM-1D communication

One or several -CVM-1D analyzers can be connected to a controller or PLC. Using this system you can operate each of the analyzers as usual as well as centralize the data in a single location. CVM-1D incorporates an RS-485 type communications output. If more than one analyzer is connected to an RS-485 serial bus, each of them must be assigned a peripheral number or address so that the communications master can send the queries regarding the different measured or calculated records to those addresses.the RS-485 connection is carried out via the shielded twisted pair communication cable, with a minimum of two wires and with a maximum distance between the communications master and the last unit of 1,200 metros. The device uses an RS-485 communications line that can support a maximum of 32 units in series per bus.
The CVM-1D type power analyzer communicates via a Modbus/RTU© protocol (Question / answer polling).

8.- Technical specification

Supply specifications :	
- Single-phase :	88...276 $V_{c.a}$
- Frequency:	50 / 60 Hz
- Maximum consumption :	2 VA
Mechanical specifications:	
- Case material :	Selft-extinguish UL94-V0 plastic
- Protection :	
Fitted unit (front panel) :	IP31
Fitted MID unit (front panel) :	IP51
Non-fitted unit (front panel):	IP20
- Maximum Dimensions (mm) :	85.5 x 64.2 x 18 mm (1 DIN rail module)
- Weight :	150 g
- Maximum cable cross-section:	10 mm^2 (6 mm^2 with end sleeve)
Environmental specifications:	
- Working temperature:	-5...+55 ºC
- Storage temperature:	-25...+70 ºC
- Humidity:	5...95% non condensing
- Maximum altitude:	2000m
Accuracy:	
- Voltage :	0.5 % ± 1 digit
- Current :	0.5 % ± 1 digit
- Power / Energy :	1 % ± 1 digit
Sensors :	
- Voltage :	Direct. Impedance 1MΩ
- Current :	Direct (shunt <0,5 mΩ)
Power factor :	0.5...1
Measurement range:	0.5...120% FS

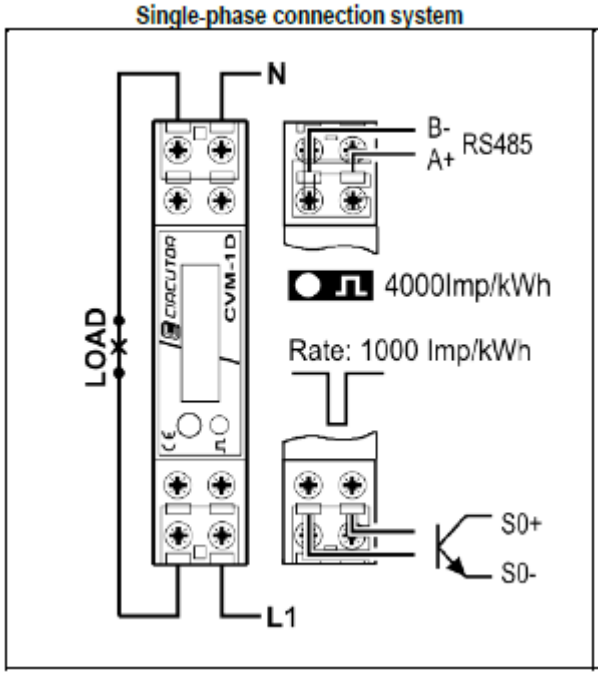
Metering circuit:	
- Nominal voltage / Tolerance:	110...230 $V_{c.a}$ / ±20 %
- Nominal voltage / MID Tolerance:	230 $V_{c.a}$ / ±20 %
- Frequency :	50 / 60Hz
- MID frequency:	50Hz
- Nominal current/minimum/maximum:	5 A / 250 mA / 32 A
- Start current (/st):	20 mA
- Reference current (/ref):	5 A
- Transition current (/tr):	500 mA
Output transistor specifications	
- Typo: opto-isolated transistor (open collector)	NPN
- Maximum operating voltage:	42 $V_{c.c.}$
- Maximum operating current:	50 mA
- Maximum frequency:	1000 imp / kW·h
- Pulse width:	40...200ms (configurable)

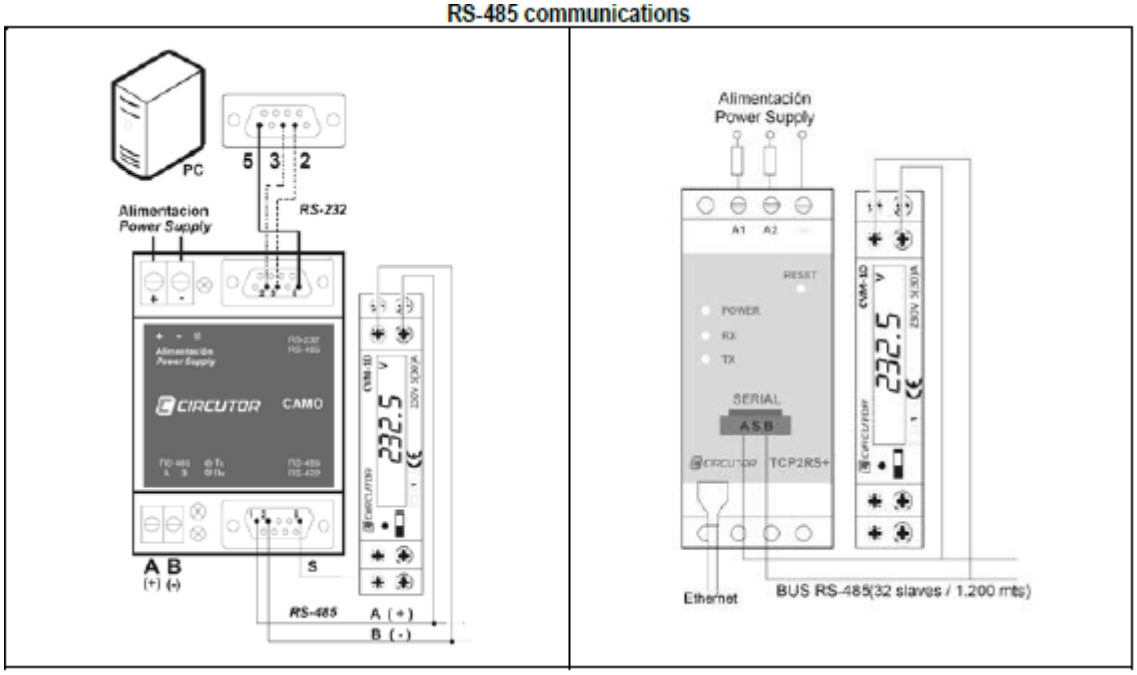
Safety:
CATIII-300 EN61010-1:2010 EN61010-2-030:2011. Double insulation. Pollution degree II. Means for disconnecting the power from the device must be provided in the installation. Wire conductor are must be choosen depending on current to flow acroos the device. Minimum recommended wire is 1mm^2

Standards :
EN 50470-1, EN50470-3, EN62053-21, EN62053-23, EN61010-1:2010, EN 61000-6-4, EN 55022

Energy meter: Class B EN50470-3 Active Energy, Class 2 EN62053-23 Reactive Energy.

9.- Connection





ANEXO II. MINIMALMODBUS

1. Typical usage

The instrument is typically connected via a serial port, and a USB-to-serial adaptor should be used on most modern computers.

For example, consider an instrument (slave) with Modbus RTU mode and address number 1 to which we are to communicate via a serial port with the name `/dev/ttyUSB1`. The instrument stores the measured temperature in register 289. For this instrument a temperature of 77.2 C is stored as (the integer) 772, why we use 1 decimal. To read this data from the instrument:

```
#!/usr/bin/env python
import minimalmodbus

instrument = minimalmodbus.Instrument('/dev/ttyUSB1', 1) # port name, slave address (in
decimal)

## Read temperature (PV = ProcessValue) ##
temperature = instrument.read_register(289, 1) # Registernumber, number of decimals
print temperature

## Change temperature setpoint (SP) ##
NEW_TEMPERATURE = 95
instrument.write_register(24, NEW_TEMPERATURE, 1) # Registernumber, value, number of
decimals for storage
```

Correspondingly for Modbus ASCII mode:

```
instrument = minimalmodbus.Instrument('/dev/ttyUSB1', 1, minimalmodbus.MODE_ASCII)
```

2. Default values

Most of the serial port parameters have the default values defined in the Modbus standard (19200 8N1):

```
instrument.serial.port          # this is the serial port name
instrument.serial.baudrate = 19200   # Baud
instrument.serial.bytesize = 8
instrument.serial.parity   = serial.PARITY_NONE
instrument.serial.stopbits = 1
instrument.serial.timeout  = 0.05   # seconds

instrument.address     # this is the slave address number
instrument.mode = minimalmodbus.MODE_RTU   # rtu or ascii mode
```

These can be overridden:

```
instrument.serial.timeout = 0.2
```

To see which settings you actually are using:

```
print instrument
```

To change the parity setting, use:

```
import serial
instrument.serial.parity = serial.PARITY_EVEN
```

3. Using multiple instruments

Use a single script for talking to all your instruments (if connected via the same serial port). Create several instrument objects like:

```
instrumentA = minimalmodbus.Instrument('/dev/ttyUSB1', 1)
instrumentB = minimalmodbus.Instrument('/dev/ttyUSB1', 2)
```

Running several scripts using the same port will give problems.

4. Handling communication errors

Your top-level code should be able to handle communication errors. This is typically done with try-except.

Instead of running:

```
print(instrument.read_register(4143))
```

Use:

```
try:
    print(instrument.read_register(4143))
except IOError:
    print("Failed to read from instrument")
```

API for MinimalModbus

MinimalModbus: A Python driver for the Modbus RTU and Modbus ASCII protocols via serial port (via RS485 or RS232).

minimalmodbus.BAUDRATE= *19200*

Default value for the baudrate in Baud (int).

minimalmodbus.PARITY= *'N'*

Default value for the parity. See the pySerial module for documentation. Defaults to serial.PARITY_NONE

minimalmodbus.BYTESIZE= *8*

Default value for the bytesize (int).

minimalmodbus.STOPBITS= *1*

Default value for the number of stopbits (int).

minimalmodbus.TIMEOUT= *0.05*

Default value for the timeout value in seconds (float).

minimalmodbus.CLOSE_PORT_AFTER_EACH_CALL= *False*

Default value for port closure setting.

class **minimalmodbus.Instrument**(*port*, *slaveaddress*, *mode='rtu'*)

Instrument class for talking to instruments (slaves) via the Modbus RTU or ASCII protocols (via RS485 or RS232).

Args:

- port (str): The serial port name, for example `/dev/ttyUSB0` (Linux), `/dev/tty.usbserial` (OS X) or `COM4` (Windows).
- slaveaddress (int): Slave address in the range 1 to 247 (use decimal numbers, not hex).
- mode (str): Mode selection. Can be MODE_RTU or MODE_ASCII.

address= *None*

Slave address (int). Most often set by the constructor (see the class documentation).

mode= *None*

Slave mode (str), can be MODE_RTU or MODE_ASCII. Most often set by the constructor (see the class documentation).

New in version 0.6.

debug= *None*

Set this to `True` to print the communication details. Defaults to `False`.

close_port_after_each_call= *None*

If this is `True`, the serial port will be closed after each call. Defaults to `CLOSE_PORT_AFTER_EACH_CALL`. To change it, set the value `minimalmodbus.CLOSE_PORT_AFTER_EACH_CALL=True` .

precalculate_read_size= *None*

If this is `False`, the serial port reads until timeout instead of just reading a specific number of bytes. Defaults to `True`.

New in version 0.5.

handle_local_echo= *None*

Set to to `True` if your RS-485 adaptor has local echo enabled. Then the transmitted message will immeadiately appear at the receive line of the RS-485 adaptor. MinimalModbus will then read and discard this data, before reading the data from the slave. Defaults to `False`.

New in version 0.7.

read_bit(*registeraddress*, *functioncode=2*)

Read one bit from the slave.

Args:

- registeraddress (int): The slave register address (use decimal numbers, not hex).
- functioncode (int): Modbus function code. Can be 1 or 2.

Returns:

The bit value 0 or 1 (int).

Raises:

ValueError, TypeError, IOError

write_bit(*registeraddress*, *value*, *functioncode=5*)

Write one bit to the slave.

Args:

- registeraddress (int): The slave register address (use decimal numbers, not hex).
- value (int): 0 or 1
- functioncode (int): Modbus function code. Can be 5 or 15.

Returns:

None

Raises:

ValueError, TypeError, IOError

`read_register`(*registeraddress*, *numberOfDecimals=0*, *functioncode=3*, *signed =False*)

Read an integer from one 16-bit register in the slave, possibly scaling it.

The slave register can hold integer values in the range 0 to 65535 (“Unsigned INT16”).

Args:

- registeraddress (int): The slave register address (use decimal numbers, not hex).
- numberOfDecimals (int): The number of decimals for content conversion.
- functioncode (int): Modbus function code. Can be 3 or 4.
- signed (bool): Whether the data should be interpreted as unsigned or signed.

If a value of 77.0 is stored internally in the slave register as 770, then use `numberOfDecimals=1` which will divide the received data by 10 before returning the value.

Similarly `numberOfDecimals=2` will divide the received data by 100 before returning the value.

Some manufacturers allow negative values for some registers. Instead of an allowed integer range 0 to 65535, a range -32768 to 32767 is allowed. This is implemented as any received value in the upper range (32768 to 65535) is interpreted as negative value (in the range -32768 to -1).

Use the parameter `signed=True` if reading from a register that can hold negative values. Then upper range data will be automatically converted into negative return values (two's complement).

`signed`	Data type in slave	Alternative name	Range
`False`	Unsigned INT16	Unsigned short	0 to 65535
`True`	INT16	Short	-32768 to 32767

Returns:

The register data in numerical value (int or float).

Raises:

ValueError, TypeError, IOError

`write_register`(*registeraddress*, *value*, *numberOfDecimals=0*, *functioncode=16*, *signed=False*)

Write an integer to one 16-bit register in the slave, possibly scaling it.

The slave register can hold integer values in the range 0 to 65535 ("Unsigned INT16").

Args:

- registeraddress (int): The slave register address (use decimal numbers, not hex).
- value (int or float): The value to store in the slave register (might be scaled before sending).
- numberOfDecimals (int): The number of decimals for content conversion.
- functioncode (int): Modbus function code. Can be 6 or 16.
- signed (bool): Whether the data should be interpreted as unsigned or signed.

To store for example `value=77.0`, use `numberOfDecimals=1` if the slave register will hold it as 770 internally. This will multiply `value` by 10 before sending it to the slave register.

Similarly `numberOfDecimals=2` will multiply `value` by 100 before sending it to the slave register.

For discussion on negative values, the range and on alternative names, see `read_register()`.

Use the parameter `signed=True` if writing to a register that can hold negative values. Then negative input will be automatically converted into upper range data (two's complement).

Returns:

None

Raises:

ValueError, TypeError, IOError

`read_long`(*registeraddress*, *functioncode=3*, *signed=False*)

Read a long integer (32 bits) from the slave.

Long integers (32 bits = 4 bytes) are stored in two consecutive 16-bit registers in the slave.

Args:

- registeraddress (int): The slave register start address (use decimal numbers, not hex).
- functioncode (int): Modbus function code. Can be 3 or 4.
- signed (bool): Whether the data should be interpreted as unsigned or signed.

`signed`	**Data type in slave**	**Alternative name**	**Range**
`False`	Unsigned INT32	Unsigned long	0 to 4294967295
`True`	INT32	Long	-2147483648 to 2147483647

Returns:

The numerical value (int).

Raises:

ValueError, TypeError, IOError

`write_long`(*registeraddress*, *value*, *signed=False*)

Write a long integer (32 bits) to the slave.

Long integers (32 bits = 4 bytes) are stored in two consecutive 16-bit registers in the slave.

Uses Modbus function code 16.

For discussion on number of bits, number of registers, the range and on alternative names, see `read_long()`.

Args:

- registeraddress (int): The slave register start address (use decimal numbers, not hex).
- value (int or long): The value to store in the slave.
- signed (bool): Whether the data should be interpreted as unsigned or signed.

Returns:

None

Raises:

ValueError, TypeError, IOError

`read_float`(*registeraddress*, *functioncode=3*, *numberOfRegisters=2*)

Read a floating point number from the slave.

Floats are stored in two or more consecutive 16-bit registers in the slave. The encoding is according to the standard IEEE 754.

There are differences in the byte order used by different manufacturers. A floating point value of 1.0 is encoded (in single precision) as 3f800000 (hex). In this implementation the data will be sent as `'\x3f\x80'` and `'\x00\x00'` to two consecutetive registers . Make sure to test that it makes sense for your

instrument. It is pretty straight-forward to change this code if some other byte order is required by anyone (see support section).

Args:

- registeraddress (int): The slave register start address (use decimal numbers, not hex).
- functioncode (int): Modbus function code. Can be 3 or 4.
- numberOfRegisters (int): The number of registers allocated for the float. Can be 2 or 4.

Type of floating point number in slave	Size	Registers	Range
Single precision (binary32)	32 bits (4 bytes)	2 registers	1.4E-45 to 3.4E38
Double precision (binary64)	64 bits (8 bytes)	4 registers	5E-324 to 1.8E308

Returns:

The numerical value (float).

Raises:

ValueError, TypeError, IOError

`write_float`(*registeraddress*, *value*, *numberOfRegisters=2*)

Write a floating point number to the slave.

Floats are stored in two or more consecutive 16-bit registers in the slave.

Uses Modbus function code 16.

For discussion on precision, number of registers and on byte order, see `read_float()`.

Args:

- registeraddress (int): The slave register start address (use decimal numbers, not hex).
- value (float or int): The value to store in the slave
- numberOfRegisters (int): The number of registers allocated for the float. Can be 2 or 4.

Returns:

None

Raises:

ValueError, TypeError, IOError

`read_string`(***registeraddress***, ***numberOfRegisters=16***, ***functioncode=3***)

Read a string from the slave.

Each 16-bit register in the slave are interpreted as two characters (1 byte = 8 bits). For example 16 consecutive registers can hold 32 characters (32 bytes).

Args:

- registeraddress (int): The slave register start address (use decimal numbers, not hex).
- numberOfRegisters (int): The number of registers allocated for the string.
- functioncode (int): Modbus function code. Can be 3 or 4.

Returns:

The string (str).

Raises:

ValueError, TypeError, IOError

`write_string`(***registeraddress***, ***textstring***, ***numberOfRegisters=16***)

Write a string to the slave.

Each 16-bit register in the slave are interpreted as two characters (1 byte = 8 bits). For example 16 consecutive registers can hold 32 characters (32 bytes).

Uses Modbus function code 16.

Args:

- registeraddress (int): The slave register start address (use decimal numbers, not hex).
- textstring (str): The string to store in the slave
- numberOfRegisters (int): The number of registers allocated for the string.

If the textstring is longer than the 2*numberOfRegisters, an error is raised. Shorter strings are padded with spaces.

Returns:

None

Raises:

ValueError, TypeError, IOError

`read_registers`(*registeraddress*, *numberOfRegisters*, *functioncode=3*)

Read integers from 16-bit registers in the slave.

The slave registers can hold integer values in the range 0 to 65535 (“Unsigned INT16”).

Args:

- registeraddress (int): The slave register start address (use decimal numbers, not hex).
- numberOfRegisters (int): The number of registers to read.
- functioncode (int): Modbus function code. Can be 3 or 4.

Any scaling of the register data, or converting it to negative number (two’s complement) must be done manually.

> **Returns:**

The register data (a list of int).

> **Raises:**

ValueError, TypeError, IOError

`write_registers`(*registeraddress*, *values*)[

Write integers to 16-bit registers in the slave.

The slave register can hold integer values in the range 0 to 65535 (“Unsigned INT16”).

Uses Modbus function code 16.

The number of registers that will be written is defined by the length of the `values` list.

> **Args:**

- registeraddress (int): The slave register start address (use decimal numbers, not hex).
- values (list of int): The values to store in the slave registers.

Any scaling of the register data, or converting it to negative number (two’s complement) must be done manually.

> **Returns:**

None

> **Raises:**

ValueError, TypeError, IOError

5. Modbus details

Modbus data types

The Modbus standard defines storage in:

- Bits
- Registers (16-bit). Can hold integers in the range 0 to 65535 (dec), which is 0 to ffff (hex). Also called ‘unsigned INT16’ or ‘unsigned short’.

Some deviations from the official standard:

Scaling of register values

Some manufacturers store a temperature value of 77.0 C as 770 in the register, to allow room for one decimal.

Negative numbers (INT16 = short)

Some manufacturers allow negative values for some registers. Instead of an allowed integer range 0-65535, a range -32768 to 32767 is allowed. This is implemented as any received value in the upper range (32768-65535) is interpreted as negative value (in the range -32768 to -1). This is two’s complement and is described at https://en.wikipedia.org/wiki/Two%27s_complement. Help functions to calculate the two’s complement value (and back) are provided in MinimalModbus.

Long integers (‘Unsigned INT32’ or ‘INT32’)

These require 32 bits, and are implemented as two consecutive 16-bit registers. The range is 0 to 4294967295, which is called ‘unsigned INT32’. Alternatively negative values can be stored if the instrument is defined that way, and is then called ‘INT32’ which has the range -2147483648 to 2147483647.

Floats (single or double precision)

Single precision floating point values (binary32) are defined by 32 bits (4 bytes), and are implemented as two consecutive 16-bit registers. Correspondingly, double precision floating point values (binary64) use 64 bits (8 bytes) and are implemented as four consecutive 16-bit registers. How to convert from the bit values to the floating point value is described in the standard IEEE 754, as seen in https://en.wikipedia.org/wiki/Floating_point. Unfortunately the byte order might differ between manufacturers of Modbus instruments.

Strings

Each register (16 bits) is interpreted as two characters (each 1 byte = 8 bits). Often 16 consecutive registers are used, allowing 32 characters in the string.

8-bit registers

For example Danfoss use 8-bit registers for storage of some settings internally in the instruments. The data is nevertheless transmitted as 16 bit over the serial link, so you can read and write like normal (but with values limited to the range 0-255).

6. Implemented functions

These are the functions to use for reading and writing registers and bits of your instrument. Study the documentation of your instrument to find which Modbus function code to use. The function codes (F code) are given in decimal in this table.

Data type in slave	Read	F code	Write	F code
Bit	`read_bit()`	2 [or 1]	`write_bit()`	5 [or 15]
Register Integer, possibly scaled	`read_register()`	3 [or 4]	`write_register()`	16 [or 6]
Long	`read_long()`	3 [or 4]	`write_long()`	16

Data type in slave	Read	F code	Write	F code
(32 bits = 2 registers)				
Float (32 or 64 bits)	`read_float()`	3 [or 4]	`write_float()`	16
String	`read_string()`	3 [or 4]	`write_string()`	16
Registers Integers	`read_registers()`	3 [or 4]	`write_registers()`	16

See the API for MinimalModbus: API for MinimalModbus.

7. Modbus implementation details

In Modbus RTU, the request message is sent from the master in this format:

- Slave address [1 Byte]
- Function code [1 Byte]. Allowed range is 1 to 127 (in decimal).
- Payload data [0 to 252 Bytes]
- CRC [2 Bytes]. It is a Cyclic Redundancy Check code, for error checking of the message

The response from the client is similar, but with other payload data.

Function code **(in decimal)**	**Payload data to slave** **(Request)**	**Payload data from slave** **(Response)**
1 Read bits (coils)	Start address [2 Bytes] Number of coils [2 Bytes]	Byte count [1 Byte] Value [k Bytes]
2 Read discrete inputs	Start address [2 Bytes] Number of inputs [2 Bytes]	Byte count [1 Byte] Value [k Bytes]
3 Read holding registers	Start address [2 Bytes] Number of registers [2 Bytes]	Byte count [1 Byte] Value [n*2 Bytes]
4 Read input registers	Start address [2 Bytes] Number of registers [2 Bytes]	Byte count [1 Byte] Value [n*2 Bytes]
5 Write single bit (coil)	Output address [2 Bytes] Value [2 Bytes]	Output address [2 Bytes] Value [2 Bytes]
6 Write single register	Register address [2 Bytes] Value [2 Bytes]	Register address [2 Bytes] Value [2 Bytes]
15 Write multiple bits (coils)	Start address [2 Bytes] Number of outputs [2 Bytes] Byte count [1 Byte] Value [k Bytes]	Start address [2 Bytes] Number of outputs [2 Bytes]

Function code (in decimal)	Payload data to slave (Request)	Payload data from slave (Response)
16 Write multiple registers	Start address [2 Bytes] Number of registers [2 Bytes] Byte count [1 Byte] Value [n*2 Bytes]	Start address [2 Bytes] Number of regist [2 Bytes]

TODO Validate

For function code 5, the only valid values are 0000 (hex) or FF00 (hex), representing OFF and ON respectively.

It is seen in the table above that the request and response messages are similar for function code 1 to 4. The same can be said about function code 5 and 6, and also about 15 and 16.

For finding how the k Bytes for the value relates to the number of registers etc (n), see the Modbus documents referred to above.

8. MODBUS ASCII format

This driver also supports Modbus ASCII mode.

Basically, a byte with value 0-255 in Modbus RTU mode will in Modbus ASCII mode be sent as two characters corresponding to the hex value of that byte.

For example a value of 76 (dec) = 4C (hex) is sent as the byte 0x4C in Modbus RTU mode. This byte happens to correspond to the character ‘L’ in the ASCII encoding. Thus for Modbus RTU this is sent: `'\x4C'`, which is a string of length 1 and will print as ‘L’.

The same value will in Modbus ASCII be sent as the string ‘4C’, which has a length of 2.

The frame format is slightly different for Modbus ASCII. The request message is sent from the master in this format:

- Start [1 character]. It is the colon (:).
- Slave Address [2 characters]
- Function code [2 characters]
- Payload data [0 to 2*252 characters]
- LRC [2 characters]. The LRC is a Longitudinal Redundancy Check code, for error checking of the message.
- Stop [2 characters]. The stop characters are carriage return (`'\r'` = `'\x0D'`) and line feed (`'\n'` = `'\x0A'`).

9. Manual testing of Modbus equipment

Look in your equipment's manual to find working communication examples.

You can make a small Python program to test the communication:

```
TODO: Change this to a RTU example

import serial
ser = serial.Serial('/dev/ttyUSB0', 19200, timeout=1)
print ser

ser.write(':010310010001EA\r\n')
print repr(ser.read(1000)) # Read 1000 bytes, or wait for timeout
```

It should print something like:

```
Serial<id=0x9faa08c, open=True>(port='/dev/ttyUSB0', baudrate=19200, bytesize=8,
parity='N', stopbits=1, timeout=1, xonxoff=False, rtscts=False, dsrdtr=False)
:0103020136C3
```

Correspondingly for Modbus ASCII, change the write command to for example:

```
TODO: Verify

ser.write(':010310010001EA\r\n')
```

It should then print something like:

```
Serial<id=0x9faa08c, open=True>(port='/dev/ttyUSB0', baudrate=19200, bytesize=8,
parity='N', stopbits=1, timeout=1, xonxoff=False, rtscts=False, dsrdtr=False)
:0103020136C3
```

It is also easy to test Modbus ASCII equipment from Linux command line. First must the appropriate serial port be set up properly:

- Print port settings: `stty -F /dev/ttyUSB0`
- Print all settings for a port: `stty -F /dev/ttyUSB0 -a`

- Reset port to default values: `stty -F /dev/ttyUSB0 sane`
- Change port to raw behavior: `stty -F /dev/ttyUSB0 raw`
- and: `stty -F /dev/ttyUSB0 -echo -echoe -echok`
- Change port baudrate: `stty -F /dev/ttyUSB0 19200`

To send out a Modbus ASCII request (read register 0x1001 on slave 1), and print out the response:

```
cat /dev/ttyUSB0 &
echo -e ":010310010001EA\r\n" > /dev/ttyUSB0
```

The reponse will be something like:

```
:0103020136C3
```

ANEXO III. XML.DOM.MINIDOM

DOM applications typically start by parsing some XML into a DOM. With **xml.dom.minidom**, this is done through the parse functions:

```
from xml.dom.minidom import parse, parseString

dom1 = parse('c:\\temp\\mydata.xml')  # parse an XML file by name

datasource = open('c:\\temp\\mydata.xml')
dom2 = parse(datasource)  # parse an open file

dom3 = parseString('<myxml>Some data<empty/> some more data</myxml>')
```

The **parse()** function can take either a filename or an open file object.

xml.dom.minidom.**parse**(*filename_or_file*[, *parser*[, *bufsize*]])

Return a **Document** from the given input. *filename_or_file* may be either a file name, or a file-like object. *parser*, if given, must be a SAX2 parser object. This function will change the document handler of the parser and activate namespace support; other parser configuration (like setting an entity resolver) must have been done in advance.

If you have XML in a string, you can use the **parseString()** function instead:

xml.dom.minidom.**parseString**(*string*[, *parser*])

Return a **Document** that represents the *string*. This method creates a **StringIO** object for the string and passes that on to **parse()**.

Both functions return a **Document** object representing the content of the document.

What the **parse()** and **parseString()** functions do is connect an XML parser with a “DOM builder” that can accept parse events from any SAX parser and convert them into a DOM tree. The name of the functions are perhaps misleading, but are easy to grasp when learning the interfaces. The parsing of the document will be completed before these functions return; it’s simply that these functions do not provide a parser implementation themselves.

You can also create a **Document** by calling a method on a “DOM Implementation” object. You can get this object either by calling the **getDOMImplementation()** function in the **xml.dom** package or the **xml.dom.minidom** module. Using the implementation from the **xml.dom.minidom** module will always return a **Document** instance from the

minidom implementation, while the version from **xml.dom** may provide an alternate implementation (this is likely if you have the PyXML package installed). Once you have a **Document**, you can add child nodes to it to populate the DOM:

```
from xml.dom.minidom import getDOMImplementation

impl = getDOMImplementation()

newdoc = impl.createDocument(None, "some_tag", None)
top_element = newdoc.documentElement
text = newdoc.createTextNode('Some textual content.')
top_element.appendChild(text)
```

Once you have a DOM document object, you can access the parts of your XML document through its properties and methods. These properties are defined in the DOM specification. The main property of the document object is the **documentElement** property. It gives you the main element in the XML document: the one that holds all others. Here is an example program:

```
dom3 = parseString("<myxml>Some data</myxml>")
assert dom3.documentElement.tagName == "myxml"
```

When you are finished with a DOM tree, you may optionally call the **unlink()** method to encourage early cleanup of the now-unneeded objects. **unlink()** is an **xml.dom.minidom**-specific extension to the DOM API that renders the node and its descendants are essentially useless. Otherwise, Python's garbage collector will eventually take care of the objects in the tree.

The W3C recommendation for the DOM supported by **xml.dom.minidom**.

1. DOM Objects

The definition of the DOM API for Python is given as part of the **xml.dom** module documentation. This section lists the differences between the API and**xml.dom.minidom**.

Node.**unlink**()

Break internal references within the DOM so that it will be garbage collected on versions of Python without cyclic GC. Even when cyclic GC is available, using this can make large amounts of memory available sooner, so calling this on DOM objects as soon as they are no longer needed is good practice. This only needs to be called on the **Document** object, but may be called on child nodes to discard children of that node.

Node.**writexml**(*writer*, *indent=""*, *addindent=""*, *newl=""*)

Write XML to the writer object. The writer should have a **`write()`** method which matches that of the file object interface. The *indent* parameter is the indentation of the current node. The *addindent* parameter is the incremental indentation to use for subnodes of the current one. The *newl* parameter specifies the string to use to terminate newlines.

For the **`Document`** node, an additional keyword argument *encoding* can be used to specify the encoding field of the XML header.

Changed in version 2.1: The optional keyword parameters *indent*, *addindent*, and *newl* were added to support pretty output.

Changed in version 2.3: For the **`Document`** node, an additional keyword argument *encoding* can be used to specify the encoding field of the XML header.

`Node.`**`toxml`**([*encoding*])

Return the XML that the DOM represents as a string.

With no argument, the XML header does not specify an encoding, and the result is Unicode string if the default encoding cannot represent all characters in the document. Encoding this string in an encoding other than UTF-8 is likely incorrect, since UTF-8 is the default encoding of XML.

With an explicit *encoding* [1] argument, the result is a byte string in the specified encoding. It is recommended that this argument is always specified. To avoid **`UnicodeError`** exceptions in case of unrepresentable text data, the encoding argument should be specified as “utf-8”.

Changed in version 2.3: the *encoding* argument was introduced; see **`writexml()`**.

`Node.`**`toprettyxml`**([*indent=""*[, *newl=""*[, *encoding=""*]]])

Return a pretty-printed version of the document. *indent* specifies the indentation string and defaults to a tabulator; *newl* specifies the string emitted at the end of each line and defaults to `\n`.

New in version 2.1.

Changed in version 2.3: the encoding argument was introduced; see **`writexml()`**.

The following standard DOM methods have special considerations with **`xml.dom.minidom`**:

`Node.`**`cloneNode`**(*deep*)

Although this method was present in the version of **xml.dom.minidom** packaged with Python 2.0, it was seriously broken. This has been corrected for subsequent releases.

2. DOM Example

This example program is a fairly realistic example of a simple program. In this particular case, we do not take much advantage of the flexibility of the DOM.

```
import xml.dom.minidom

document = """\
<slideshow>
<title>Demo slideshow</title>
<slide><title>Slide title</title>
<point>This is a demo</point>
<point>Of a program for processing slides</point>
</slide>

<slide><title>Another demo slide</title>
<point>It is important</point>
<point>To have more than</point>
<point>one slide</point>
</slide>
</slideshow>
"""

dom = xml.dom.minidom.parseString(document)

def getText(nodelist):
    rc = []
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            rc.append(node.data)
    return ''.join(rc)

def handleSlideshow(slideshow):
    print "<html>"
    handleSlideshowTitle(slideshow.getElementsByTagName("title")[0])
    slides = slideshow.getElementsByTagName("slide")
    handleToc(slides)
    handleSlides(slides)
    print "</html>"

def handleSlides(slides):
    for slide in slides:
        handleSlide(slide)

def handleSlide(slide):
    handleSlideTitle(slide.getElementsByTagName("title")[0])
    handlePoints(slide.getElementsByTagName("point"))

def handleSlideshowTitle(title):
    print "<title>%s</title>" % getText(title.childNodes)

def handleSlideTitle(title):
```

```
    print "<h2>%s</h2>" % getText(title.childNodes)

def handlePoints(points):
    print "<ul>"
    for point in points:
        handlePoint(point)
    print "</ul>"

def handlePoint(point):
    print "<li>%s</li>" % getText(point.childNodes)

def handleToc(slides):
    for slide in slides:
        title = slide.getElementsByTagName("title")[0]
        print "<p>%s</p>" % getText(title.childNodes)

handleSlideshow(dom)
```

3. minidom and the DOM standard

The **xml.dom.minidom** module is essentially a DOM 1.0-compatible DOM with some DOM 2 features (primarily namespace features).

Usage of the DOM interface in Python is straight-forward. The following mapping rules apply:

- Interfaces are accessed through instance objects. Applications should not instantiate the classes themselves; they should use the creator functions available on the **Document** object. Derived interfaces support all operations (and attributes) from the base interfaces, plus any new operations.
- Operations are used as methods. Since the DOM uses only **in** parameters, the arguments are passed in normal order (from left to right). There are no optional arguments. `void` operations return `None`.
- IDL attributes map to instance attributes. For compatibility with the OMG IDL language mapping for Python, an attribute `foo` can also be accessed through accessor methods **_get_foo()** and **_set_foo()**. `readonly` attributes must not be changed; this is not enforced at runtime.
- The types `short int`, `unsigned int`, `unsigned long long`, and `boolean` all map to Python integer objects.
- The type `DOMString` maps to Python strings. **xml.dom.minidom** supports either byte or Unicode strings, but will normally produce Unicode strings. Values of type `DOMString` may also be `None` where allowed to have the IDL `null` value by the DOM specification from the W3C.
- `const` declarations map to variables in their respective scope (e.g. `xml.dom.minidom.Node.PROCESSING_INSTRUCTION_NODE`); they must not be changed.

- `DOMException` is currently not supported in **`xml.dom.minidom`**. Instead, **`xml.dom.minidom`** uses standard Python exceptions such as **`TypeError`** and **`AttributeError`**.
- **`NodeList`** objects are implemented using Python's built-in list type. Starting with Python 2.2, these objects provide the interface defined in the DOM specification, but with earlier versions of Python they do not support the official API. They are, however, much more "Pythonic" than the interface defined in the W3C recommendations.

ANEXO IV. FTPLIB

1. ftplib – FTP Protocol Client

This module defines the class **FTP** and a few related items. The **FTP** class implements the client side of the FTP protocol. You can use this to write Python programs that perform a variety of automated FTP jobs, such as mirroring other FTP servers. It is also used by the module **urllib** to handle URLs that use FTP. For more information on FTP (File Transfer Protocol), see Internet **RFC 959**.

Here's a sample session using the **ftplib** module:

```
>>> from ftplib import FTP
>>> ftp = FTP('ftp.debian.org')     # connect to host, default port
>>> ftp.login()                     # user anonymous, passwd anonymous@
'230 Login successful.'
>>> ftp.cwd('debian')               # change into "debian" directory
>>> ftp.retrlines('LIST')           # list directory contents
-rw-rw-r--    1 1176     1176         1063 Jun 15 10:18 README
...
drwxr-sr-x    5 1176     1176         4096 Dec 19  2000 pool
drwxr-sr-x    4 1176     1176         4096 Nov 17  2008 project
drwxr-xr-x    3 1176     1176         4096 Oct 10  2012 tools
'226 Directory send OK.'
>>> ftp.retrbinary('RETR README', open('README', 'wb').write)
'226 Transfer complete.'
>>> ftp.quit()
```

The module defines the following items:

class `ftplib.`**`FTP`**([*host*[, *user*[, *passwd*[, *acct*[, *timeout*]]]]])

Return a new instance of the **FTP** class. When *host* is given, the method call `connect(host)` is made. When *user* is given, additionally the method call `login(user, passwd, acct)` is made (where *passwd* and *acct* default to the empty string when not given). The optional *timeout* parameter specifies a timeout in seconds for blocking operations like the connection attempt (if is not specified, the global default timeout setting will be used).

Changed in version 2.6: timeout was added.

class `ftplib.`**`FTP_TLS`**([*host*[, *user*[, *passwd*[, *acct*[, *keyfile*[, *certfile*[, *context*[, *timeout*]]]]]]]])

A **FTP** subclass which adds TLS support to FTP as described in **RFC 4217**. Connect as usual to port 21 implicitly securing the FTP control connection before authenticating. Securing the data connection requires the user to

explicitly ask for it by calling the **prot_p()** method. *context* is a **ssl.SSLContext** object which allows bundling SSL configuration options, certificates and private keys into a single (potentially long-lived) structure. Please read Security considerations for best practices.

keyfile and *certfile* are a legacy alternative to *context* – they can point to PEM-formatted private key and certificate chain files (respectively) for the SSL connection.

New in version 2.7.

Changed in version 2.7.10: The *context* parameter was added.

Here's a sample session using the **FTP_TLS** class:

```
>>> from ftplib import FTP_TLS
>>> ftps = FTP_TLS('ftp.python.org')
>>> ftps.login()           # login anonymously before securing control
channel
>>> ftps.prot_p()          # switch to secure data connection
>>> ftps.retrlines('LIST') # list directory content securely
total 9
drwxr-xr-x   8 root     wheel        1024 Jan  3  1994 .
drwxr-xr-x   8 root     wheel        1024 Jan  3  1994 ..
drwxr-xr-x   2 root     wheel        1024 Jan  3  1994 bin
drwxr-xr-x   2 root     wheel        1024 Jan  3  1994 etc
d-wxrwxr-x   2 ftp      wheel        1024 Sep  5 13:43 incoming
drwxr-xr-x   2 root     wheel        1024 Nov 17  1993 lib
drwxr-xr-x   6 1094     wheel        1024 Sep 13 19:07 pub
drwxr-xr-x   3 root     wheel        1024 Jan  3  1994 usr
-rw-r--r--   1 root     root          312 Aug  1  1994 welcome.msg
'226 Transfer complete.'
>>> ftps.quit()
>>>
```

exception ftplib.**error_reply**

Exception raised when an unexpected reply is received from the server.

exception ftplib.**error_temp**

Exception raised when an error code signifying a temporary error (response codes in the range 400–499) is received.

exception ftplib.**error_perm**

Exception raised when an error code signifying a permanent error (response codes in the range 500–599) is received.

exception ftplib.**error_proto**

Exception raised when a reply is received from the server that does not fit the response specifications of the File Transfer Protocol, i.e. begin with a digit in the range 1–5.

ftplib.**all_errors**
The set of all exceptions (as a tuple) that methods of **FTP** instances may raise as a result of problems with the FTP connection (as opposed to programming errors made by the caller). This set includes the four exceptions listed above as well as **socket.error** and **IOError**.

2. FTP Objects

Several methods are available in two flavors: one for handling text files and another for binary files. These are named for the command which is used followed by `lines` for the text version or `binary` for the binary version.

FTP instances have the following methods:

FTP.**set_debuglevel**(*level*)
Set the instance's debugging level. This controls the amount of debugging output printed. The default, `0`, produces no debugging output. A value of`1` produces a moderate amount of debugging output, generally a single line per request. A value of `2` or higher produces the maximum amount of debugging output, logging each line sent and received on the control connection.

FTP.**connect**(*host*[, *port*[, *timeout*]])
Connect to the given host and port. The default port number is `21`, as specified by the FTP protocol specification. It is rarely needed to specify a different port number. This function should be called only once for each instance; it should not be called at all if a host was given when the instance was created. All other methods can only be used after a connection has been made.

The optional *timeout* parameter specifies a timeout in seconds for the connection attempt. If no *timeout* is passed, the global default timeout setting will be used.

Changed in version 2.6: timeout was added.

FTP.**getwelcome**()
Return the welcome message sent by the server in reply to the initial connection. (This message sometimes contains disclaimers or help information that may be relevant to the user.)

FTP.**login**([*user*[, *passwd*[, *acct*]]])
Log in as the given *user*. The *passwd* and *acct* parameters are optional and default to the empty string. If no *user* is specified, it defaults to`'anonymous'`. If *user* is `'anonymous'`, the default *passwd* is `'anonymous@'`. This function

should be called only once for each instance, after a connection has been established; it should not be called at all if a host and user were given when the instance was created. Most FTP commands are only allowed after the client has logged in. The *acct* parameter supplies “accounting information”; few systems implement this.

`FTP.`**`abort`**()

Abort a file transfer that is in progress. Using this does not always work, but it’s worth a try.

`FTP.`**`sendcmd`**(*command*)

Send a simple command string to the server and return the response string.

`FTP.`**`voidcmd`**(*command*)

Send a simple command string to the server and handle the response. Return nothing if a response code corresponding to success (codes in the range 200–299) is received. Raise **`error_reply`** otherwise.

`FTP.`**`retrbinary`**(*command*, *callback*[, *maxblocksize*[, *rest*]])

Retrieve a file in binary transfer mode. *command* should be an appropriate `RETR` command: `'RETR filename'`. The *callback* function is called for each block of data received, with a single string argument giving the data block. The optional *maxblocksize* argument specifies the maximum chunk size to read on the low-level socket object created to do the actual transfer (which will also be the largest size of the data blocks passed to *callback*). A reasonable default is chosen. *rest* means the same thing as in the **`transfercmd()`** method.

`FTP.`**`retrlines`**(*command*[, *callback*])

Retrieve a file or directory listing in ASCII transfer mode. *command* should be an appropriate `RETR` command (see **`retrbinary()`**) or a command such as `LIST`, `NLST` or `MLSD` (usually just the string `'LIST'`). `LIST` retrieves a list of files and information about those files. `NLST` retrieves a list of file names. On some servers, `MLSD` retrieves a machine readable list of files and information about those files. The *callback* function is called for each line with a string argument containing the line with the trailing CRLF stripped. The default *callback* prints the line to `sys.stdout`.

`FTP.`**`set_pasv`**(*val*)

Enable “passive” mode if *val* is true, otherwise disable passive mode. (In Python 2.0 and before, passive mode was off by default; in Python 2.1 and later, it is on by default.)

`FTP.`**`storbinary`**(*command*, *fp*[, *blocksize*, *callback*, *rest*])

Store a file in binary transfer mode. *command* should be an appropriate `STOR` command: `"STOR filename"`. *fp* is an open file object which is read until EOF using its **`read()`** method in blocks of size *blocksize* to provide the data to be stored. The *blocksize* argument defaults to 8192. *callback* is an optional single parameter callable that is called on each block of data after it is sent. *rest* means the same thing as in the **`transfercmd()`** method.

Changed in version 2.1: default for *blocksize* added.

Changed in version 2.6: callback parameter added.

Changed in version 2.7: rest parameter added.

`FTP.`**`storlines`**(*command*, *fp*[, *callback*])

Store a file in ASCII transfer mode. *command* should be an appropriate `STOR` command (see **`storbinary()`**). Lines are read until EOF from the open file object *fp* using its **`readline()`** method to provide the data to be stored. *callback* is an optional single parameter callable that is called on each line after it is sent.

Changed in version 2.6: callback parameter added.

`FTP.`**`transfercmd`**(*cmd*[, *rest*])

Initiate a transfer over the data connection. If the transfer is active, send an `EPRT` or `PORT` command and the transfer command specified by *cmd*, and accept the connection. If the server is passive, send an `EPSV` or `PASV` command, connect to it, and start the transfer command. Either way, return the socket for the connection.

If optional *rest* is given, a `REST` command is sent to the server, passing *rest* as an argument. *rest* is usually a byte offset into the requested file, telling the server to restart sending the file's bytes at the requested offset, skipping over the initial bytes. Note however that RFC 959 requires only that *rest*be a string containing characters in the printable range from ASCII code 33 to ASCII code 126. The **`transfercmd()`** method, therefore, converts *rest* to a string, but no check is performed on the string's contents. If the server does not recognize the `REST` command, an **`error_reply`** exception will be raised. If this happens, simply call **`transfercmd()`** without a *rest* argument.

`FTP.`**`ntransfercmd`**(*cmd*[, *rest*])

Like **`transfercmd()`**, but returns a tuple of the data connection and the expected size of the data. If the expected size could not be computed, `None` will be returned as the expected size. *cmd* and *rest* means the same thing as in **`transfercmd()`**.

`FTP.`**`nlst`**(*argument*[, *...*])

Return a list of file names as returned by the `NLST` command. The optional *argument* is a directory to list (default is the current server directory). Multiple arguments can be used to pass non-standard options to the `NLST` command.

`FTP.`**`dir`**(*argument*[, *...*])

Produce a directory listing as returned by the `LIST` command, printing it to standard output. The optional *argument* is a directory to list (default is the current server directory). Multiple arguments can be used to pass non-standard options to the `LIST` command. If the last argument is a function, it is used as a *callback* function as for **`retrlines()`**; the default prints to `sys.stdout`. This method returns `None`.

`FTP.`**`rename`**(*fromname*, *toname*)

Rename file *fromname* on the server to *toname*.

`FTP.`**`delete`**(*filename*)

Remove the file named *filename* from the server. If successful, returns the text of the response, otherwise raises **`error_perm`** on permission errors or**`error_reply`** on other errors.

`FTP.`**`cwd`**(*pathname*)

Set the current directory on the server.

`FTP.`**`mkd`**(*pathname*)

Create a new directory on the server.

`FTP.`**`pwd`**()

Return the pathname of the current directory on the server.

`FTP.`**`rmd`**(*dirname*)

Remove the directory named *dirname* on the server.

`FTP.`**`size`**(*filename*)

Request the size of the file named *filename* on the server. On success, the size of the file is returned as an integer, otherwise `None` is returned. Note that

the `SIZE` command is not standardized, but is supported by many common server implementations.

`FTP.`**`quit`**()

Send a `QUIT` command to the server and close the connection. This is the “polite” way to close a connection, but it may raise an exception if the server responds with an error to the `QUIT` command. This implies a call to the **`close()`** method which renders the **`FTP`** instance useless for subsequent calls (see below).

`FTP.`**`close`**()

Close the connection unilaterally. This should not be applied to an already closed connection such as after a successful call to **`quit()`**. After this call the **`FTP`** instance should not be used any more (after a call to **`close()`** or **`quit()`** you cannot reopen the connection by issuing another **`login()`** method).

3. FTP_TLS Objects

`FTP_TLS` class inherits from **`FTP`**, defining these additional objects:

`FTP_TLS.`**`ssl_version`**

The SSL version to use (defaults to **`ssl.PROTOCOL_SSLv23`**).

`FTP_TLS.`**`auth`**()

Set up secure control connection by using TLS or SSL, depending on what specified in **`ssl_version()`** attribute.

`FTP_TLS.`**`prot_p`**()

Set up secure data connection.

`FTP_TLS.`**`prot_c`**()

Set up clear text data connection.

ANEXO V. CÓDIGOS

1. Código principal Raspberry (CODIGOFINAL.py)

Importación de los módulos necesarios:

```
#!/usr/bin/python
import minimalmodbus
import ftplib
import time
from xml.dom import minidom
import usb
import os
```

Ilustración A.V.1.1

El módulo **time** es necesario para obtener la fecha y la hora posteriormente en el código.

El módulo **usb** es necesario para detectar los dispositivos conectados a los puertos USB y reconocer si está conectado el USB to RS-485.

El módulo **os** se utiliza para acceder a funcionalidades dependientes del sistema operativo. Permitirá manipular archivos y directorios para lectura y escritura.

El módulo **minimalmodbus** será el encargado de extraer los bits de información del bus de datos y disponerlos en formato decimal.

El módulo **ftplib** permitirá manejar la comunicación FTP entre el dispositivo y el servidor remoto. (Python Software Foundation, 2017)

El módulo **xml.dom.minidom** es necesario para construir el documento XML que se envía por FTP.

Definición del protocolo de datos:

```
#-------------DEFINICION DE PUERTO----------------------------------------
        instrument = minimalmodbus.Instrument('/dev/ttyUSB0', 1)
        instrument.serial.baudrate = 9600
        instrument.serial.bytesize = 8
        instrument.serial.parity = minimalmodbus.serial.PARITY_NONE
        instrument.serial.stopbits = 1
        instrument.serial.timeout = 2
        instrument.debug = False
        minimalmodbus.close_port_after_each_call=True
        instrument.mode = minimalmodbus.MODE_RTU
```

Ilustración A.V.1.2

Esta configuración se realiza acorde con la definición de protocolo MODBUS/RTU (***Ilustración 5.1.1***). La velocidad de transmisión por defecto es 9600 baudios, se puede cambiar en el Circutor (ver **Anexo I**). La instrucción **serial.timeout** establece un tiempo de espera a la respuesta de la petición de datos. En este caso con 2 segundos es suficiente. Es necesario cerrar el puerto después de cada llamada para un correcto funcionamiento con la instrucción **.close_port_after_each_call**. Para más detalles ver **Anexo II**.

Recolección de datos:

```
#-------------RECOLECCION DE DATOS--------------------------------------------------------------------
arrayDatos=[float(instrument.read_long(0x0000, 4))/10.,float(instrument.read_long(0x0032, 4))/10.,
float(instrument.read_long(0x0044, 4))/10.,float(instrument.read_long(0x0002, 4))/100.,
float(instrument.read_long(0x0034, 4))/100.,float(instrument.read_long(0x0046, 4))/100.,
float(instrument.read_long(0x0004, 4, True))/100.,float(instrument.read_long(0x0036, 4, True))/100.,
float(instrument.read_long(0x0048,4, True))/100.,float(instrument.read_long(0x0006, 4,True))/100.,
float(instrument.read_long(0x0038, 4, True))/100.,float(instrument.read_long(0x004A,4, True))/100.,
float(instrument.read_long(0x0008, 4,True))/100., float(instrument.read_long(0x003A, 4, True))/100.,
float(instrument.read_long(0x004C,4, True))/100., float(instrument.read_long(0x000A, 4,True))/100.,
float(instrument.read_long(0x003C, 4, True))/100., float(instrument.read_long(0x004E,4, True))/100.,
float(instrument.read_long(0x000C, 4,True))/100., float(instrument.read_long(0x003E, 4, True))/100.,
float(instrument.read_long(0x0050,4, True))/100., float(instrument.read_long(0x000E,3,True))/100.,
float(instrument.read_long(0x0040, 3, False))/100., float(instrument.read_long(0x0052,3,True))-float(256.),
float(instrument.read_long(0x0010, 4,False))/100., float(instrument.read_long(0x0042, 4, False))/100.,
float(instrument.read_long(0x0054,4, False))/100., float(instrument.read_long(0x0012, 4,False))/100.,
float(instrument.read_long(0x0014, 4,False))/100., float(instrument.read_long(0x0016, 4,False))/100.,
float(instrument.read_long(0x0018, 4,False))/100., float(instrument.read_long(0x001A, 4,False))/100.,
float(instrument.read_long(0x001C, 4,False))/100., float(instrument.read_long(0x001E, 4,False))/100.,
float(instrument.read_long(0x0020, 4,False))/100., float(instrument.read_long(0x0022, 4,False))/100.,
float(instrument.read_long(0x0028, 4,False))/100., float(instrument.read_long(0x002A, 4,False))/100.,
float(instrument.read_long(0x002C, 4,False))/100., float(instrument.read_long(0x002E, 4,False))/100.,
float(instrument.read_long(0x0030, 4,False))/100.]
```

Ilustración A.V.1.3

Para recoger los datos se utiliza la instrucción **read_long()** que se aplica al instrumento creado para definir la transmisión de datos anteriormente (***Ilustración A.V.1.2***). Esta instrucción requiere los argumentos de dirección en el mapa de memoria, function code y como parámetro opcional, el signo. La instrucción quedará así:

read_long(*dirección, function code, signed*)

Algunas medidas hay que dividirlas por 100 o por 10 para que la coma esté en su lugar según se muestra en los mapas de memoria de Circutor (**Anexo I**).

Para mostrar las medidas con decimales se transforman en tipo float mediante la instrucción **float()**.

Todos los datos se almacenan en **arrayDatos**.

A continuación se muestran los arrays de nombres y unidades para posteriormente enlazarlos con cada medida.

```
arrayNombres=['Voltaje instantaneo: ', 'Voltaje maximo: ', 'Voltaje minimo: ', 'Corriente instantanea: ',
 'Corriente maxima: ', 'Corriente minima: ', 'Potencia Activa instantanea: ', 'Potencia Activa maxima: ',
 'Potencia Activa minima: ', 'Potencia Reactiva(L/C) instantanea: ', 'Potencia Reactiva(L/C) maxima: ',
 'Potencia Reactiva(L/C) minima: ', 'Potencia Inductiva Reactiva instantanea: ',
 'Potencia Inductiva Reactiva maxima: ', 'Potencia Inductiva Reactiva minima: ',
 'Potencia Capacitiva Reactiva instantanea: ', 'Potencia Capacitiva Reactiva maxima: ',
 'Potencia Capacitiva Reactiva minima: ', 'Potencia Aparente instantanea: ', 'Potencia Aparente maxima: ',
 'Potencia Aparente minima: ', 'Factor de Potencia instantaneo: ', 'Factor de Potencia maximo: ',
 'Factor de Potencia minimo: ', 'Demanda de Potencia instantanea: ', 'Demanda de Potencia maxima: ',
 'Demanda de Potencia minima: ', 'Energia Activa instantanea: ',
 'Energia Inductiva Reactiva instantanea: ', 'Energia Capacitiva Reactiva instantanea: ',
 'Energia Reactiva (L/C) instantanea: ', 'Energia Activa Parcial instantanea: ',
 'Energia Inductiva Reactiva Parcial instantanea: ', 'Energia Capacitiva Reactiva Parcial instantanea: ',
 'Energia Reactiva (L/C) Parcial instantanea: ', 'Energia Activa Generada instantanea: ',
 'Energia Inductiva Reactiva Generada instantanea: ', 'Energia Capacitiva Reactiva Generada instantanea: ',
 'Energia Reactiva Total (L/C) Generada instantanea: ',  'Energia Activa Parcial Generada instantanea: ',
 'Energia Inductiva Reactiva Parcial Generada instantanea: ',
 'Energia Capacitiva Reactiva Generada instantanea: ',  'Energia Total Reactiva Parcial Generada instantanea: ']
arrayUnidades=['V','V','V','A','A','A','kW','kW','kW','kvar','kvar','kvar','kvarL','kvarL','kvarL','kvarC',
'kvarC','kvarC','kVA','kVA','kVA',' ',' ',' ','kW','kW','kW','kWh','kvarLh','kvarCh','kvarh','kWh', 'kvarLh',
'kvarCh','kvarh','kWh','kvarLh','kvarCh','kvarh','kWh','kvarLh','kvarCh','kvarCh']

instrument.serial.close()
```

Ilustración A.V.1.4

Los nombres y unidades de las medidas se han transcrito de los mapas de memoria de Circutor. Es necesario que arrayDatos, arrayNombres y arrayUnidades tengan el mismo número de elementos para que el ciclo **for** que los enlaza funcione correctamente.

Una vez recogidos los datos en la instrucción ***Ilustración A.V.3*** se puede cerrar el puerto con la instrucción **instrument.serial.close()**.

Impresión de datos en XML:

```
1  #--------------IMPRESION DE DATOS XML-----------------------------------------
2  b=0
3  root = minidom.Document()
4  xml = root.createElement('documento')
5  xml.setAttribute('cliente', '88888888L')
6  root.appendChild(xml)
7  productChild=root.createElement('fecha')
8
9  productChild.setAttribute('dia',str(time.strftime("%d")))
10 productChild.setAttribute('mes',str(time.strftime("%m")))
11 productChild.setAttribute('ano',str(time.strftime("%Y")))
12
13 xml.appendChild(productChild)
14
15 productChild2=root.createElement('medidas')
16 productChild2.setAttribute('hora', str(time.strftime("%H:%M:%S")))
17 productChild.appendChild(productChild2)
18
19
20 for i in arrayDatos:
21     productChild3=root.createElement('medicion'+' nombre='+'"'+arrayNombresl[b]
22     +'"'+' unidad='+'"'+arrayUnidadesl[b]+'"'+' valor='+'"'+str(i)+'"')
23     productChild2.appendChild(productChild3)
24     b=b+1
25
26 xml.appendChild(productChild)
27
28 with open("/home/pi/"+name+".xml", "w") as f:
29     f.write(root.toprettyxml(indent="\t"))
```

Ilustración A.V.1.5

Explicación detallada del código:

Línea 3. Se crea un documento en la variable root mediante el objeto definido en el módulo **xml.dom.Document**.

Línea 4. Se crea el elemento llamado **documento** en el apartado root.

Línea 5. Se establece el atributo de nombre **cliente** y de valor **88888888L** al elemento.

Línea 6. Se introduce el elemento **xml** como subelemento de **root**.

Líneas 7-11. Se crea el elemento **productChild** y se le da 3 atributos de fecha.

Línea 13. Se introduce el elemento **productChild** como subelemento de **xml**.

Líneas 15-17. Se crea el elemento **productChild2** llamado **medidas**, con atributo ‘hora’ y se introduce como subelemento de **productChild**.

Líneas 20-23. Se crea el elemento **productChild3** donde se construye cada medida cogiendo los datos almacenados en **arrayDatos**, **arrayUnidades** y **arrayNombres** y se introduce como subelemento de **productChild2**.

Líneas 28-29. Cada vez que se ejecuta el código se abre el archivo en modo escritura (**‘w’**) de forma que se sobreescribirá el contenido existente en él. Se escribe en el fichero la estructura formada en root con la herramienta **toprettyxml()** para organizar el código XML de una forma más visual y estructurada. El nombre del archivo se obtiene de la variable **name**, que se establecerá al principio del código asignándole de nombre la fecha en formato dd-mm-YYYY. (GitHub, Wordpress, 2017)

El fichero XML resultante tiene un aspecto como este:

```xml
<?xml version="1.0" ?>
<documento cliente="88888888L">
    <fecha ano="2017" dia="21" mes="05">
        <medidas hora="11:21:47">
            <medicion nombre="Voltaje instantaneo" unidad="V" valor="232.5"/>
            <medicion nombre="Voltaje maximo" unidad="V" valor="236.2"/>
            <medicion nombre="Voltaje minimo" unidad="V" valor="223.4"/>
            <medicion nombre="Corriente instantanea" unidad="A" valor="0.0"/>
            <medicion nombre="Corriente maximo" unidad="A" valor="0.1"/>
            <medicion nombre="Corriente minima" unidad="A" valor="0.04"/>
            <medicion nombre="Potencia Activa instantanea" unidad="kW" valor="0.0"/>
            <medicion nombre="Potencia Activa maxima" unidad="kW" valor="0.0"/>
            <medicion nombre="Potencia Activa minima" unidad="kW" valor="0.0"/>
            <medicion nombre="Potencia Reactiva(L/C) instantanea" unidad="kvar" valor="0.0"/>
            <medicion nombre="Potencia Reactiva(L/C) maxima" unidad="kvar" valor="0.0"/>
            <medicion nombre="Potencia Reactiva(L/C) minima" unidad="kvar" valor="0.0"/>
            <medicion nombre="Potencia Inductiva Reactiva instantanea" unidad="kvarL" valor="0.0"/>
            <medicion nombre="Potencia Inductiva Reactiva maxima" unidad="kvarL" valor="0.0"/>
            <medicion nombre="Potencia Inductiva Reactiva minima" unidad="kvarL" valor="0.0"/>
            <medicion nombre="Potencia Capacitiva Reactiva instantanea" unidad="kvarC" valor="0.0"/>
            <medicion nombre="Potencia Capacitiva Reactiva maxima" unidad="kvarC" valor="0.0"/>
            <medicion nombre="Potencia Capacitiva Reactiva minima" unidad="kvarC" valor="0.0"/>
            <medicion nombre="Potencia Aparente instantanea" unidad="kVA" valor="0.0"/>
            <medicion nombre="Potencia Aparente maxima" unidad="kVA" valor="0.0"/>
            <medicion nombre="Potencia Aparente minima" unidad="kVA" valor="0.0"/>
            <medicion nombre="Factor de Potencia instantaneo" unidad=" " valor="1.0"/>
            <medicion nombre="Factor de Potencia maximo" unidad=" " valor="1.0"/>
            <medicion nombre="Factor de Potencia minimo" unidad=" " valor="0.0"/>
            <medicion nombre="Demanda de Potencia instantanea" unidad="kW" valor="0.0"/>
            <medicion nombre="Demanda de Potencia maxima" unidad="kW" valor="0.0"/>
            <medicion nombre="Demanda de Potencia minima" unidad="kW" valor="0.0"/>
            <medicion nombre="Energia Activa instantanea" unidad="kWh" valor="6.33"/>
            <medicion nombre="Energia Inductiva Reactiva instantanea" unidad="kvarLh" valor="0.02"/>
            <medicion nombre="Energia Capacitiva Reactiva instantanea" unidad="kvarCh" valor="0.4"/>
            <medicion nombre="Energia Reactiva (L/C) instantanea" unidad="kvarh" valor="0.42"/>
            <medicion nombre="Energia Activa Parcial instantanea" unidad="kWh" valor="0.12"/>
            <medicion nombre="Energia Inductiva Reactiva Parcial instantanea" unidad="kvarLh" valor="0.01"/>
            <medicion nombre="Energia Capacitiva Reactiva Parcial instantanea" unidad="kvarCh" valor="0.06"/>
            <medicion nombre="Energia Reactiva (L/C) Parcial instantanea" unidad="kvarh" valor="0.06"/>
            <medicion nombre="Energia Activa Generada instantanea" unidad="kWh" valor="13.47"/>
            <medicion nombre="Energia Inductiva Reactiva Generada instantanea" unidad="kvarLh" valor="0.0"/>
            <medicion nombre="Energia Capacitiva Reactiva Generada instantanea" unidad="kvarCh" valor="0.9"/>
            <medicion nombre="Energia Reactiva Total (L/C) Generada instantanea" unidad="kvarh" valor="0.9"/>
            <medicion nombre="Energia Activa Parcial Generada instantanea" unidad="kWh" valor="0.0"/>
            <medicion nombre="Energia Inductiva Reactiva Parcial Generada instantanea" unidad="kvarLh" valor="0.0"/>
            <medicion nombre="Energia Capacitiva Reactiva Generada instantanea" unidad="kvarCh" valor="0.0"/>
            <medicion nombre="Energia Total Reactiva Parcial Generada instantanea" unidad="kvarCh" valor="0.0"/>
        </medidas>
    </fecha>
</documento>
```

Ilustración A.V.1.6

Envío de datos:

```
1  #--------------ENVIO DE DATOS A SERVIDOR----------------------------------------
2  session = ftplib.FTP('11.222.333.444','syy00001@red.ujaen.es','test1234')
3  file = open('/home/pi/'+name+'.xml','rb')
4  file2 = open('/home/pi/EVENTOS.txt','rb')
5  try:
6      session.storbinary('STOR /doc/88888888L/'+name+'/'+name+str(d)+'.xml', file)
7      session.storbinary('STOR /doc/88888888L/'+name+'/EVENTOS.txt', file2)
8
9  except:
10     d=0
11     session.mkd('/doc/88888888L/'+name)
12     session.storbinary('STOR /doc/88888888L/'+name+'/'+name+str(d)+'.xml', file)
13     session.storbinary('STOR /doc/88888888L/'+name+'/EVENTOS.txt', file2)
14 d+=1
15 file.close()
16 file2.close()
17 session.quit()
18 time sleep(120)
19 infile = open('/home/pi/'+name+'.xml', 'r')
20 print(infile.read())
21 infile.close()
```

Ilustración A.V.1.7

Explicación detallada del código:

Línea 2. Se crea una sesión con el módulo **ftplib** (ver **Anexo IV**). Introduciendo características necesarias para acceder al servidor remoto: IP, usuario y contraseña.

Líneas 3-4. Se abren los archivos de medidas y eventos en modo de lectura binaria (**'rb'**).

Líneas 5-14: Se intenta guardar el archivo una carpeta con la fecha de la toma de medidas. Si la carpeta no existe porque es la primera medida que se toma en el día o porque se acaba de iniciar el aparato, se crea la carpeta y se guardan los archivos con un número **d** al final del nombre del fichero de medidas. Esto se hace para que se guarden todos los ficheros de medidas en lugar de sobreescribirse. Al final, se incrementa **d**.

Líneas 15-17: Se cierran ambos ficheros y se cierra la sesión FTP.

Línea 18. Se establece un tiempo de espera entre cada toma de medidas de 2 minutos.

Líneas 19-21. Se abre en modo lectura el fichero XML creado y se muestra su contenido en el terminal. Estas líneas de código no serán necesarias a la hora de poner en funcionamiento el dispositivo, simplemente sirven para observar el fichero final sin necesidad de abrirlo.

Para que las medidas se tomen periódicamente es necesario englobar los procesos de recolección, impresión y envío de datos en un ciclo sin fin (while 1). Así, la instrucción de espera de 2 minutos se producirá en cada ciclo.

2. Error USB desconectado

Código de detección del error en **CODIGOFINAL.py**. (Python Software Foundation, 2017)

```
1   h=1
2   while 1:
3       a=0
4       f=0
5       e=1
6       name=str(time.strftime("%d-%m-%Y"))
7       busses = usb.busses()
8       for bus in busses:
9           devices = bus.devices
10          for dev in devices:
11              if dev.idProduct==29987:
12                  a=1
13                  h=1
14                  break
15      if a==1:
16
17      #-------------DEFINICION DE PUERTO-------------------------------------------------------------------
18      #-------------RECOLECCION DE DATOS-------------------------------------------------------------------
19      #-------------IMPRESION DE DATOS XML-----------------------------------------------------------------
20      #-------------ENVIO DE DATOS A SERVIDOR--------------------------------------------------------------
21      else:
22          if h==1:
23              try:
24                  open('/home/pi/EVENTOS.txt', 'r').read()
25                  outfile = open('/home/pi/EVENTOS.txt', 'a')
26                  outfile.write('\n'+str(time.strftime("%d/%m/%Y , %H:%M:%S"))+' : El USB ha sido desconectado')
27                  outfile.close()
28
29              except:
30                  with open("/home/pi/EVENTOS.txt", "w") as f:
31                      f.write(str(time.strftime("%d/%m/%Y , %H:%M:%S"))+' : El USB ha sido desconectado')
32              h=0
```

Ilustración A.V.2.1

Explicación del código:

Líneas 2-5: Se engloba el código en un ciclo while sin fin y se reinician los valores por defecto de **a, f** y **e** en cada ciclo para el funcionamiento correcto del código.

Líneas 6: Se establece el nombre del archivo de medidas para que se actualice al inicio de cada ciclo.

Líneas 7-14: Se recopila la ID de cada dispositivo insertado en la Raspberry y se compara con la ID del RS-485 to USB utilizado (**29987**). Si hay una coincidencia, el programa detiene el rastreo de dispositivos. (GitHub, GitHub, 2017)

Líneas 15-21: Cuando el programa de rastreo de dispositivos encuentra el USB, entra en la parte de definición de puerto, recolección, guardado y envío de datos. Si el USB está desconectado, el programa entra en la instrucción **else.**

Líneas 22-31: Cuando no se encuentra el dispositivo, se realiza la prueba de abrir el archivo de **EVENTOS.txt** donde se almacenan los errores. Si se abre, se escribe la fecha y hora del error después de un salto de línea seguidos de '**El USB ha sido desconectado**'. Si es la primera vez que se ejecuta el programa y no se encuentra el fichero de **EVENTOS.txt**, se crea y se escribe el mismo mensaje.

Líneas 32: La variable **h** se utiliza para que , una vez escrito el mensaje de error, el programa no vuelva a escribir dicho mensaje hasta que se reestablezca la conexión y vuelva a fallar. De lo contrario, el código estaría escribiendo el mismo mensaje de error hasta que se conectara el USB.

3. Error Circutor inaccesible

```
1  #--------------RECOLECCION DE DATOS-----------------------------------------------------------------------------
2  while e==1:
3      try:
4          arrayDatos=[float(instrument.read_long(0x0000, 4))/10.,float(instrument.read_long(0x0032, 4))/10.,
5          float(instrument.read_long(0x0044, 4))/10.,float(instrument.read_long(0x0002, 4))/100.,.........]
6
7          if g==0:
8              try:
9                  open('/home/pi/EVENTOS.txt', 'r').read()
10                 outfile = open('/home/pi/EVENTOS.txt', 'a')
11                 outfile.write('\n'+str(time.strftime("%d/%m/%Y , %H:%M:%S"))+' : Recepcion correcta de datos')
12                 outfile.close()
13
14             except:
15                 with open("/home/pi/EVENTOS.txt", "w") as f:
16                     f.write(str(time.strftime("%d/%m/%Y , %H:%M:%S"))+' : Recepcion correcta de datos')
17             g=1
18         f=0
19         e=0
20     except:
21         g=0
22         if f==0:
23             try:
24                 open('/home/pi/EVENTOS.txt', 'r').read()
25                 outfile = open('/home/pi/EVENTOS.txt', 'a')
26                 outfile.write('\n'+'ERROR '+str(time.strftime("%d/%m/%Y , %H:%M:%S"))+
27                 ': El Circutor ha sido apagado o desconectado del USB')
28                 outfile.close()
29
30
31             except:
32                 with open("/home/pi/EVENTOS.txt", "w") as f:
33                     f.write('ERROR '+str(time.strftime("%d/%m/%Y , %H:%M:%S"))
34                     +' : El Circutor ha sido apagado o desconectado del USB')
35             f=1
```

Ilustración A.V.3.1

Explicación del código:

Línea 2: Se engloba el proceso de Recolección de datos en un segundo ciclo **while** para que el programa se quede dentro y no llegue a la instrucción de envío de datos en caso de que no haya recepción de datos.

Líneas 3-5: Se realiza la prueba de recolectar datos mediante la instrucción de la ***Ilustración 5.1.2.***

Líneas 7-12: Si se cumple que **g=0** (La primera vez que se ejecuta el código se establece esta condición)**,** se intenta abrir el fichero de **EVENTOS.txt** para escribir la fecha y hora seguidos del mensaje '**Recepción correcta de datos**'.

Líneas 14-16: En caso de no existir el fichero de **EVENTOS.txt**, se crea y se escribe el mismo mensaje.

Línea 17: Al entrar en el **if g==0** es necesario establecer la variable **g** a **1** para que no escriba el mismo mensaje siempre que entre en la recolección de datos.

Líneas 18-19: Si no hay error en el intento de recolección de datos, no será necesario seguir en el ciclo **while e==1** y se podrá continuar con las demás fases del proceso. Se establece **f=0** para que el código pueda volver a entrar en el mensaje de error en caso de que provenga de él. Si hay error, no llegará a la instrucción **e=0** y continuará en el ciclo.

Líneas 20-21: Si no se consigue respuesta en el proceso de recolección de datos, lo primero que se hará es volver a poner **g=0** para que, después de escribir el error en el recibimiento de datos, el programa pueda volver a entrar en el **if g==0** y notificar la recepción correcta de datos.

Líneas 22: El objetivo de la instrucción **if f==0** es pasar por el código de notificación del error sólo una vez, es decir, si se sigue produciendo el mismo error no volver escribir su mensaje en el fichero de **EVENTOS.txt**.

Líneas 23-35: Se intenta abrir el fichero de **EVENTOS.txt** para escribir el mensaje '**El Circutor ha sido apagado o desconectado del USB'**. Si no se puede abrir se crea el fichero y se escribe el mismo mensaje. Al final, se establece que **f=1** para no repetir el error hasta que se vuelva a realizar una conexión correcta.

4. Error Servidor inaccesible

```
1  while j==1:
2      try:
3          #--------------ENVIO DE DATOS A SERVIDOR-------------------------------------------------------------------
4          j=0
5      except:
6          if k==1:
7              try:
8                  open('/home/pi/EVENTOS.txt', 'r').read()
9                  outfile = open('/home/pi/EVENTOS.txt', 'a')
10                 outfile.write('\n'+'ERROR '+str(time.strftime("%d/%m/%Y , %H:%M:%S"))+': Servidor inaccesible')
11                 outfile.close()
12             except:
13                 with open("/home/pi/EVENTOS.txt", "w") as f:
14                     f.write('ERROR '+str(time.strftime("%d/%m/%Y , %H:%M:%S"))+' : Servidor inaccesible')
15         k=0
```

Ilustración A.V.4.1

Explicación del código:

Líneas 1-4: El código entra en un ciclo **while** e intenta hacer el envío de datos. Si el envío de datos es exitoso, se ejecutará la línea **j=0** y el código saldrá del ciclo while.

Líneas 5-14: Si se produce algún fallo al acceder al servidor o al enviar los ficheros, se escribirá el error **'Servidor inaccesible'** junto con la fecha y la hora en el fichero **EVENTOS.txt**. En caso de que el fichero no esté creado, se creará y se escribirá el mismo error.

Línea 15: Se utiliza la variable **k** para que no se escriba repetidas veces el mismo error. Esta variable se pondrá a su valor por defecto al inicio del ciclo sin fin principal para que el error se pueda volver a escribir en caso de que vuelva a ocurrir.

En Python son muy importantes las tabulaciones. El código siguiente se ha pegado con las tabulaciones necesarias para que funcione correctamente.

Contenido del archivo **CODIGOFINAL.py**:

```
#!/usr/bin/python
import minimalmodbus
import ftplib
import time
from xml.dom import minidom
import usb
import os

c=0
d=0
g=0
h=1
j=1
k=1
time.sleep(30)
while 1:
	a=0
	f=0
	e=1
	name=str(time.strftime("%d-%m-%Y"))
	busses = usb.busses()
	for bus in busses:
		devices = bus.devices
		for dev in devices:
			if dev.idProduct==29987:
				a=1
				h=1
				break
	if a==1:

#--------------DEFINICION DE PUERTO--------------------------------------------------------------------
		instrument = minimalmodbus.Instrument('/dev/ttyUSB0', 1)
		instrument.serial.baudrate = 9600
		instrument.serial.bytesize = 8
		instrument.serial.parity = minimalmodbus.serial.PARITY_NONE
		instrument.serial.stopbits = 1
		instrument.serial.timeout = 2
		instrument.debug = False
		minimalmodbus.close_port_after_each_call=True
		instrument.mode = minimalmodbus.MODE_RTU


#--------------RECOLECCION DE DATOS--------------------------------------------------------------------
		while e==1:
			try:
				arrayDatos=[float(instrument.read_long(0x0000, 4))/10.,
			float(instrument.read_long(0x0032, 4))/10.,
			float(instrument.read_long(0x0044, 4))/10.,
			float(instrument.read_long(0x0002, 4))/100.,
			float(instrument.read_long(0x0034, 4))/100.,
			float(instrument.read_long(0x0046, 4))/100.,
			float(instrument.read_long(0x0004, 4, True))/100.,
			float(instrument.read_long(0x0036, 4, True))/100.,
			float(instrument.read_long(0x0048,4, True))/100.,
			float(instrument.read_long(0x0006, 4,True))/100.,
			float(instrument.read_long(0x0038, 4, True))/100.,
			float(instrument.read_long(0x004A,4, True))/100.,
			float(instrument.read_long(0x0008, 4,True))/100.,
			float(instrument.read_long(0x003A, 4, True))/100.,
```

```
float(instrument.read_long(0x004C,4, True))/100.,
float(instrument.read_long(0x000A, 4,True))/100.,
float(instrument.read_long(0x003C, 4, True))/100.,
float(instrument.read_long(0x004E,4, True))/100.,
float(instrument.read_long(0x000C, 4,True))/100.,
float(instrument.read_long(0x003E, 4, True))/100.,
float(instrument.read_long(0x0050,4, True))/100.,
float(instrument.read_long(0x000E,3,True))/100.,
float(instrument.read_long(0x0040, 3, False))/100.,
float(instrument.read_long(0x0052,3,True))-float(256.),
float(instrument.read_long(0x0010, 4,False))/100.,
float(instrument.read_long(0x0042, 4, False))/100.,
float(instrument.read_long(0x0054,4, False))/100.,
float(instrument.read_long(0x0012, 4,False))/100.,
float(instrument.read_long(0x0014, 4,False))/100.,
float(instrument.read_long(0x0016, 4,False))/100.,
float(instrument.read_long(0x0018, 4,False))/100.,
float(instrument.read_long(0x001A, 4,False))/100.,
float(instrument.read_long(0x001C, 4,False))/100.,
float(instrument.read_long(0x001E, 4,False))/100.,
float(instrument.read_long(0x0020, 4,False))/100.,
float(instrument.read_long(0x0022, 4,False))/100.,
float(instrument.read_long(0x0028, 4,False))/100.,
float(instrument.read_long(0x002A, 4,False))/100.,
float(instrument.read_long(0x002C, 4,False))/100.,
float(instrument.read_long(0x002E, 4,False))/100.,
float(instrument.read_long(0x0030, 4,False))/100.]

        if g==0:
            try:
                open('/home/pi/EVENTOS.txt', 'r').read()
                outfile = open('/home/pi/EVENTOS.txt', 'a')
                    outfile.write('\n'+str(time.strftime("%d/%m/%Y ,
                    %H:%M:%S"))+' : Recepcion correcta de datos')
                outfile.close()

            except:
                with open("/home/pi/EVENTOS.txt", "w") as f:
                    f.write(str(time.strftime("%d/%m/%Y ,
                    %H:%M:%S"))+' : Recepcion correcta de datos')
            g=1
        e=0
        f=0

except:
        g=0
        if f==0:
            try:
                open('/home/pi/EVENTOS.txt', 'r').read()
                outfile = open('/home/pi/EVENTOS.txt', 'a')
                outfile.write('\n'+'ERROR '+str(time.strftime("%d/%m/%Y
                , %H:%M:%S"))+': El Circutor ha sido apagado o
                desconectado del USB')
                outfile.close()


            except:
                with open("/home/pi/EVENTOS.txt", "w") as f:
```

```
                                        f.write('ERROR '+str(time.strftime("%d/%m/%Y ,
                                        %H:%M:%S"))+' : El Circutor ha sido apagado o
                                        desconectado del USB')
                                f=1

#--------------IMPRESION DE DATOS XML-----------------------------------------------------------
                arrayNombres1=["Voltaje instantaneo", "Voltaje maximo", "Voltaje minimo", "Corriente
                instantanea", "Corriente maximo", "Corriente minima", "Potencia Activa instantanea",
                "Potencia Activa maxima", "Potencia Activa minima", "Potencia Reactiva(L/C) instantanea",
                "Potencia Reactiva(L/C) maxima", "Potencia Reactiva(L/C) minima", "Potencia Inductiva
                Reactiva instantanea", "Potencia Inductiva Reactiva maxima", "Potencia Inductiva Reactiva
                minima", "Potencia Capacitiva Reactiva instantanea", "Potencia Capacitiva Reactiva maxima",
                "Potencia Capacitiva Reactiva minima", "Potencia Aparente instantanea", "Potencia Aparente
                maxima", "Potencia Aparente minima", "Factor de Potencia instantaneo", "Factor de Potencia
                maximo","Factor de Potencia minimo", "Demanda de Potencia instantanea", "Demanda de
                Potencia maxima", "Demanda de Potencia minima", "Energia Activa instantanea", "Energia
                Inductiva Reactiva instantanea", "Energia Capacitiva Reactiva instantanea", "Energia Reactiva
                (L/C) instantanea", "Energia Activa Parcial instantanea", "Energia Inductiva Reactiva Parcial
                instantanea", "Energia Capacitiva Reactiva Parcial instantanea", "Energia Reactiva (L/C)
                Parcial instantanea", "Energia Activa Generada instantanea", "Energia Inductiva Reactiva
                Generada instantanea", "Energia Capacitiva Reactiva Generada instantanea", "Energia
                Reactiva Total (L/C) Generada instantanea",  "Energia Activa Parcial Generada instantanea",
                "Energia Inductiva Reactiva Parcial Generada instantanea",  "Energia Capacitiva Reactiva
                Generada instantanea",  "Energia Total Reactiva Parcial Generada instantanea"]

                arrayUnidades1=["V","V","V","A","A","A","kW","kW","kW","kvar","kvar","kvar","kvarL","
                kvarL","kvarL","kvarC","kvarC","kvarC","kVA","kVA","kVA"," "," "," ", "kW" ,"kW" ,"kW"
                , "kWh", "kvarLh","kvarCh","kvarh","kWh", "kvarLh" ,"kvarCh", "kvarh","kWh",
                "kvarLh", "kvarCh", "kvarh", "kWh","kvarLh","kvarCh","kvarCh"]
                b=0
                root = minidom.Document()
                xml = root.createElement('documento')
                xml.setAttribute('cliente', '88888888L')
                root.appendChild(xml)
                productChild=root.createElement('fecha')

                productChild.setAttribute('dia',str(time.strftime("%d")))
                productChild.setAttribute('mes',str(time.strftime("%m")))
                productChild.setAttribute('ano',str(time.strftime("%Y")))

                xml.appendChild(productChild)

                productChild2=root.createElement('medidas')
                productChild2.setAttribute('hora', str(time.strftime("%H:%M:%S")))
                productChild.appendChild(productChild2)


                for i in arrayDatos:
                        productChild3=root.createElement('medicion'+' nombre='+'"'+arrayNombres1[b]
                        +'"'+' unidad='+'"'+arrayUnidades1[b]+'"'+' valor='+'"'+str(i)+'")
                        productChild2.appendChild(productChild3)
                        b=b+1

                xml.appendChild(productChild)

                with open("/home/pi/"+name+".xml", "w") as f:
                        f.write(root.toprettyxml(indent="\t"))
```

```
#--------------ENVIO DE DATOS A SERVIDOR-------------------------------------------------------------------------

                infile = open('/home/pi/'+name+'.xml', 'r')
                print(infile.read())
                infile.close()

                while j==1:
                        try:

                                session = ftplib.FTP('11.222.333.444','syy00001@red.ujaen.es','test1234')
                                file = open('/home/pi/'+name+'.xml','rb')
                                file2 = open('/home/pi/EVENTOS.txt','rb')
                                try:
                                        session.storbinary('STOR
                                        /doc/88888888L/'+name+'/'+name+str(d)+'.xml', file)
                                        session.storbinary('STOR /doc/88888888L/'+name+'/EVENTOS.txt',
                                        file2)

                                except:
                                        d=0
                                        session.mkd('/doc/88888888L/'+name)
                                        session.storbinary('STOR
                                        /doc/88888888L/'+name+'/'+name+str(d)+'.xml', file)
                                        session.storbinary('STOR /doc/88888888L/'+name+'/EVENTOS.txt',
                                        file2)
                                d+=1
                                j=0
                                file.close()
                                file2.close()
                                session.quit()
                        except:
                                if k==1:
                                        try:
                                                open('/home/pi/EVENTOS.txt', 'r').read()
                                                outfile = open('/home/pi/EVENTOS.txt', 'a')
                                                outfile.write('\n'+'ERROR '+str(time.strftime("%d/%m/%Y
                                                , %H:%M:%S"))+': Servidor inaccesible')
                                                outfile.close()
                                        except:
                                                with open("/home/pi/EVENTOS.txt", "w") as f:
                                                        f.write('ERROR '+str(time.strftime("%d/%m/%Y ,
                                                        %H:%M:%S"))+' : Servidor inaccesible')
                                k=0


                time.sleep(120)
        else:
                if h==1:
                        try:
                                open('/home/pi/EVENTOS.txt', 'r').read()
                                outfile = open('/home/pi/EVENTOS.txt', 'a')
                                outfile.write('\n'+str(time.strftime("%d/%m/%Y , %H:%M:%S"))+' : El USB
                                ha sido desconectado')
                                outfile.close()

                        except:
                                with open("/home/pi/EVENTOS.txt", "w") as f:
                                        f.write(str(time.strftime("%d/%m/%Y , %H:%M:%S"))+' : El USB
                                        ha sido desconectado')
```

```
                    h=0
name2=str(time.strftime("%d-%m-%Y"))
if name!=name2:

            os.remove("/home/pi/"+name+".xml")
```

5. Códigos Android Studio (activity_main.xml)

Para crear el layout de la visión web en Android Studio se edita el archivo **activity_main.xml** de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="right"
    android:weightSum="1">

    <WebView
        android:id="@+id/webView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"></WebView>


</LinearLayout>
```

Ilustración A.V.5.1

El layout principal es el que viene por defecto en la aplicación y se utiliza como fondo para colocar encima el visionado web.

Se añade un layout WebView dentro del layout principal de forma que se ajuste a los márgenes y se establece una id para el WebView para posteriormente utilizarlo en la actividad principal.

Otro archivo que es necesario editar es el **AndroidManifest.xml**. Este archivo contiene configuraciones más generales como el icono de la aplicación o sus permisos.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sergio.medidas">
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="MEDIDAS"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Ilustración A.V.5.2

Se conceden permisos para acceder a internet y para obtener información de las redes. Se elige el icono ic_launcher previamente descargado como archivo de imagen y se establece el nombre “MEDIDAS” a la aplicación.

Para añadir un botón de refrescar en la barra de acción de la aplicación hay que crear un menú. Para ello, se crea la carpeta **menu** dentro de la carpeta **res** con un fichero **main_activity_actions.xml** con el botón del menu.

```xml
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/add"
        android:icon="@android:drawable/ic_popup_sync"
        android:title="@string/add"
        android:orderInCategory="1"
        app:showAsAction="always"/>
</menu>
```

Ilustración A.V.5.3

Se establece como identificador **add** y se elige el icono del botón **ic_popyp_sync**. Se ha establecido el orden del botón a 1 ya que es el único que hay y se mostrará siempre debido a la instrucción **showAsAction=”always”**.

Es necesario dar un nombre al botón, en este caso se llamará “Refresh”. Para ello se añade al fichero **strings.xml** el identificador del botón junto a su nombre.

```
1  <resources>
2      <string name="app_name">MEDIDAS</string>
3      <string name="add">Refresh</string>
4  </resources>
```

Ilustración A.V.5.4

Una vez configurado el aspecto de la aplicación se puede programar sus acciones en el archivo **MainActivity.java.**

```
1  package com.example.sergio.medidas;
2  import android.support.v7.app.AppCompatActivity;
3  import android.os.Bundle;
4  import android.view.Menu;
5  import android.view.MenuInflater;
6  import android.view.MenuItem;
7  import android.webkit.WebView;
8  import android.webkit.WebViewClient;
9
10
11 public class MainActivity extends AppCompatActivity {
12
13     @Override
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_main);
17
18
19         WebView myWebView = (WebView) this.findViewById(R.id.webView);
20         myWebView.setWebViewClient(new WebViewClient());
21         myWebView.loadUrl("https://dev.code21.net/electricas/11042017-BACKEND-ADMINISTRACION/");
22
23
24         myWebView.getSettings().setJavaScriptEnabled(true);
25     }
26
27
28
29     @Override
30     public boolean onCreateOptionsMenu(Menu menu) {
31         MenuInflater inflater=getMenuInflater();
32         inflater.inflate(R.menu.main_activity_actions, menu);
33         return true;
34     }
```

Ilustración A.V.5.5

Explicación del código:

Líneas 1-8: Se importan los archivos necesarios para utilizar las variables externas al archivo MainActivity.java.

Líneas 11-25: En la clase principal se crea un método llamado **OnCreate** para que aparezca la visión establecida en **activity_main.xml** al abrir la aplicación. Se crea

un objeto de WebView para que se visualice una págiana web concreta en el layout creado en **activity_main.xml.** La web que se visualiza es la misma donde se visualizarán las medidas desde un PC.

Líneas 30-33: Se crea un método **OnCreateOpcionsMenu** para visualizar el botón de refrescar al abrir la aplicación.

```
37      @Override
38      public boolean onOptionsItemSelected(MenuItem item) {
39
40          WebView myWebView = (WebView) this.findViewById(R.id.webView);
41          String url=myWebView.getUrl();
42          myWebView.loadUrl( url );
43
44          return super.onOptionsItemSelected(item);
45      }
46  }
```

Ilustración A.V.5.6

Líneas 38-44: Se crea el método onOpcionsItemSelected para dar una acción al botón de refrescar. En este caso, se vuelve a cargar la URL en la que se encuentra la aplicación al pulsar el botón.

6. Código servidor remoto (lecturas.php)

```php
<?php

$xmlDoc = new DOMDocument();

$files = glob('upload/doc/88888888L/*');
usort($files, function($a, $b) {
    return filemtime($a) > filemtime($b);
});

$files1 = glob(end($files)."/*.xml");
usort($files1, function($a, $b) {
    return filemtime($a) > filemtime($b);
});

$xmlDoc->load( end($files1) );

    $documento = $xmlDoc->getElementsByTagName( "documento" );
    $fecha = $xmlDoc->getElementsByTagName( "fecha" );
    $hora = $xmlDoc->getElementsByTagName( "medidas" );
    $medicion = $xmlDoc->getElementsByTagName( "medicion" );

foreach($documento as $element)
{
    $dniCliente = $element->getAttribute('cliente');
}

```

Ilustración A.V.6.1

Explicación del código:

Línea 3: Se crea un documento en la variable **xmlDoc**. (Cowburn, 2017)

Líneas 5-8: Se escoge el directorio del DNI del cliente y se ordenan los archivos que hay en el interior por orden numérico. En el interior los nombres de los archivos son fechas.

Línas 10-13: Se escoge el último directorio de los ordenados en la instrucción anterior (fecha más reciente) y se ordenan los archivos XML de su interior de igual manera.

Líneas 15-20: Se carga el fichero XML más reciente en la variable xmlDoc y se extraen del mismo los elementos **documento, fecha, hora, medicion**.

Líneas 22-25: Se rastrea el elemento **documento** para encontrar el atributo **cliente**.

```
27  foreach($fecha as $element)
28  {
29      $dia = $element->getAttribute('dia');
30      $mes = $element->getAttribute('mes');
31      $ano = $element->getAttribute('ano');
32  }
33
34  foreach($hora as $element)
35  {
36      $time = $element->getAttribute('hora');
37  }
38
39          echo "Cliente: ".$dniCliente."<br>";
40          echo "Fecha: ".$dia."/".$mes."/".$ano."<br>";
41          echo "Hora: ".$time."<br>"."<br>";
42
43  foreach($medicion as $element)
44  {
45      $nombre = $element->getAttribute('nombre');
46      $unidad = $element->getAttribute('unidad');
47      $valor = $element->getAttribute('valor');
48
49          echo $nombre.": ".$valor.$unidad."<br>";
50  }
51  ?>
```

Ilustración A.V.6.2

Líneas 27-32: Se rastrea el elemento **fecha** para extraer los atributos **dia, mes** y **ano**.

Líneas 34-37: Se rastrea el elemento **hora** para extraer el atributo **hora**.

Líneas 39-41: Se empieza visualizando en la web el DNI del cliente, la fecha y la hora. Todos seguidos de un salto de línea.

Líneas 43-47: Del el elemento **medicion**, se extraen los atributos **nombre, unidad** y **valor** de las medidas.

Líneas 49-50: Cada vez que se encuentre un elemento de **medicion** se visualizarán en la web sus atributos.

GLOSARIO

Metrología. Ciencia que tiene por objeto el estudio de los sistemas de pesas y medidas.

SSH. Secure SHell.

CPU. Central Processing Unit.

GPU. Graphics Processor Unit.

USB. Universal Serial Bus.

HDMI. High Definition Multimedia Interface.

SD. Secure Digital.

GPIO. General Purpose Input/Output.

LCD. Liquid Cristal Display.

RTU. Remote Terminal Unit.

LAN. Local Area Network.

NTP. Network Time Protocol.

Dongle. Pequeña pieza de hardware que se conecta a otro dispositivo para darle una funcionalidad adicional.

ID. Interruptor Diferencial.

ICP. Interruptor de Control de Potencia.

IG. Interruptor General.

RTC. Real Time Clock.

Script. Archivo de órdenes o procesamiento por lotes.

I2C. Inter-Integrated Circuit.

IP. Internet Protocol.

DHCP. Dynamic Host Configuration Protocol.

Inet addr. Internet Address.

SSID. Service Set Identifier.

XML. Extensible Markup Language.

FTP. File Transfer Protocol.

P2P. Peer-to-Peer.

Long. Tipo de dato que usa 4 bytes de información. Tiene un rango de -2147483648 a 2147483647.

Float. Tipo de dato utilizado para representar hasta 6 decimales. Tiene un rango de $-3{,}4 \cdot 10^{38}$ a $3{,}4 \cdot 10^{38}$.

APK. Android Application Package.

Placa base. Tarjeta de circuito impreso a la que se conectan los componentes que constituyen la computadora u ordenador.

Apache. Servidor web HTTP de código abierto.

TLS/SSL. Transport Layer Security. Secure Sockets Layer. Seguridad de la capa de transporte. Capa de puertos seguros.

SCP. Secure Copy Protocol.

SFTP. SSH File Transfer Protocol.

BIBLIOGRAFÍA

Berg, J. (2016). *MinimalModbus*. Obtenido de
https://minimalmodbus.readthedocs.io/en/master/
(fecha de último acceso 21 de Junio de 2017)

Castillo, J. E. (2012). *Historia de la metrología*. Obtenido de
https://jegcgarcas.wordpress.com/2012/04/24/historia-de-la-metrologia/
(fecha de último acceso 21 de Junio de 2017)

CIRCUTOR, S. (2017). *Serie CVM-1D*. Obtenido de
http://circutor.es/es/productos/medida-y-control/analizadores-de-redes-fijos/analizadores-de-redes/serie-cvm-1d-detail
(fecha de último acceso 21 de Junio de 2017)

Cowburn, P. (2017). *Manual de PHP*. Obtenido de
http://php.net/manual/es/index.php
(fecha de último acceso 21 de Junio de 2017)

drewkeller. (2014). *drewkeller.com*. Obtenido de
http://www.drewkeller.com/blog/adding-hardware-clock-raspberry-pi-ds3231
(fecha de último acceso 21 de Junio de 2017)

ESTESO, M. P. (2016). *TUTORIAL RASPBERRY PI*. Obtenido de
https://geekytheory.com/tutorial-raspberry-pi-configurar-wif/
(fecha de último acceso 21 de Junio de 2017)

EXES. (2016). *Manual de XML*. Obtenido de
http://www.mundolinux.info/que-es-xml.htm
(fecha de último acceso 21 de Junio de 2017)

Foundation Python Software. (2017). *Minimal DOM implementation*. Obtenido de
https://docs.python.org/2/library/xml.dom.minidom.html
(fecha de último acceso 21 de Junio de 2017)

GitHub. (2016). *GitHub*. Obtenido de
https://github.com/walac/pyusb/blob/master/docs/tutorial.rst
(fecha de último acceso 21 de Junio de 2017)

GitHub. (2013). *Wordpress*. Obtenido de
https://amatellanes.wordpress.com/2013/05/06/lectura-y-escritura-de-ficheros-en-python/
(fecha de último acceso 21 de Junio de 2017)

Handkraft. (2016). *AC/DC 5V 600mA Fuente de Alimentación Conmutada*. Obtenido de
https://www.amazon.es/Alimentaci%C3%B3n-Conmutada-Incorporada-Descubierto-Convertidor/dp/B01M3YVVU0/ref=pd_sbs_147_2?_encoding=UTF8&psc=1&refRID=VVYBXW9N4FBKFVJVD3HH
(fecha de último acceso 21 de Junio de 2017)

Mario G., B. (2016). *frambuesapi.co*. Obtenido de
http://www.frambuesapi.co/2013/09/25/tutorial-5-conexion-remota-al-raspberry-pi-usando-ssh/
(fecha de último acceso 21 de Junio de 2017)

MATT. (2014). *Simple Guide to the RPi GPIO Header and Pins*. Obtenido de
http://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/
(fecha de último acceso 21 de Junio de 2017)

Piña, J. B. (2012). *Tecnología de Transmisión RS485*. Obtenido de
http://uhu.es/antonio.barragan/content/311-tecnologia-transmision-rs485
(fecha de último acceso 21 de Junio de 2017)

Python Software Foundation. (2017). Obtenido de
https://docs.python.org/2/library/ftplib.html
(fecha de último acceso 21 de Junio de 2017)

Python Software Foundation. (2017). Obtenido de
https://docs.python.org/3/tutorial/errors.html
(fecha de último acceso 21 de Junio de 2017)

Python Software Foundation. (2014). *MinimalModbus*. Obtenido de
https://pypi.python.org/pypi/MinimalModbus/0.6
(fecha de último acceso 21 de Junio de 2017)

RAE, R. A. (2017). *Diccionario de la lengua española*. Obtenido de
http://dle.rae.es/?id=Om9ZDVF
(fecha de último acceso 21 de Junio de 2017)

RASPBERRY PI FOUNDATION. (2014). *RASPBERRY PI 1 MODEL B+*. Obtenido de https://www.raspberrypi.org/products/model-b-plus/ (fecha de último acceso 21 de Junio de 2017)

Raspipc.es, E. (2015). *Añadir un Reloj RTC a la Raspberry Pi*. Obtenido de http://raspipc.es/blog/?p=161 (fecha de último acceso 21 de Junio de 2017)

SODIAL(R). (2014). *Amazon*. Obtenido de https://www.amazon.es/SODIAL-Convertidor-Adaptador-USB-485-Soporta/dp/B00K67XKVI/ref=sr_1_4/260-8017730-2090365? (fecha de último acceso 21 de Junio de 2017)

Ventura, V. (2015). *Reloj en tiempo real DS3231 con comunicaciones I2C*. Obtenido de https://polaridad.es/rtc-reloj-tiempo-real-ds3231-comunicaciones-i2c/ (fecha de último acceso 21 de Junio de 2017)