

LIKINDOY - Documentación del proyecto

Juan Miguel Taboada Godoy (juanmi@likindoy.org)
Juan José Denis Corrales (jjdenis@axaragua.com)

Versión del documento **20080703**

y con uso intensivo de los protocolos y tecnologías popularizados en Internet.

Este documento trata de describir los aspectos básicos de Likindoy y orientar al usuario en la filosofía de trabajo y uso del mismo.

Preámbulo

Likindoy es un proyecto que tiene por objeto la creación de un sistema de control y de adquisición de datos (SCADA) a partir de software libre, ordenadores personales (PC)

Índice

1. Introducción	3	3. Manuales	12
1.1. Qué es Likindoy	3	3.1. Manual de usuario	12
1.2. Qué es un SCADA	4	3.2. Manual de operador	13
1.3. Más concretamente: Qué hace Likindoy. . .	4	3.2.1. Preguntas antes de comenzar	13
1.3.1. Ejemplo: Gráficas diarias de proceso.	4	3.2.2. La adquisición de datos	15
1.3.2. Ejemplo: Visualización de datos en tiempo real.	4	3.2.3. La sencillez del cargador	15
1.3.3. Otras aplicaciones.	4	3.2.4. Los misterios del senales.ods	16
1.4. ¿Para qué se crea Likindoy?	5	3.2.5. Introducción a la generación de gráficas	18
1.4.1. Gráficas de proceso de gran calidad.	5	3.2.6. Obtener los resultados por email	22
1.4.2. SCADA libre.	5	3.2.7. La temida RTU	23
1.4.3. SCADA en PC, no en PLC.	7	3.2.8. Un adelanto de la nueva RTU	30
1.5. Quién ha desarrollado Likindoy.	7	3.2.9. El servidor UDP	31
1.6. Sistema de desarrollo.	7	3.2.10. El sistema de estadísticas de red	32
1.7. Evolución temporal.	7	3.3. Manual de administrador	33
2. Instalación del sistema	9	3.3.1. Debuging del programa	33
2.1. Requisitos	9	3.3.2. Usando el cron	34
2.2. Instalación automática	10	3.3.3. Conocer el estado actual	34
2.3. Instalación manual	10	3.3.4. El funcionamiento	34
2.4. Cargando los primeros datos	11	3.3.5. Desarrolladores de Likindoy	35
2.5. Generando las primeras gráficas	11	4. Detalles del sistema	35
2.6. Registrando los primeros datos	12	4.1. Análisis funcional	35
		4.1.1. Alcance funcional	35
		4.1.2. Casos de éxito	36
		4.1.3. Advertencia	37
		4.2. Diseño técnico	38

4.2.1.	Diagrama principal del sistema . . .	38
4.2.2.	Detalles del árbol	38
4.2.3.	Lenguajes de programación usados	41
4.3.	Modelos de datos	42
4.3.1.	El fichero de señales	42
4.3.2.	Configuraciones	42
4.3.3.	Ficheros PESYR-INFOCEL	43
4.3.4.	Ficheros SIGNALS	43
4.3.5.	Bases de datos SQL	44
5.	Doxygen	44
6.	Licencia	45
7.	Conclusiones	45

1. Introducción

1.1. Qué es Likindoy

Likindoy es un **SCADA** libre, fuertemente basado en protocolos y tecnologías popularizadas en Internet. Intenta ser un sistema simple, fácil de entender por «hackers» e ingenieros de control.

Sus principales características son:

- Es libre. Por lo que evita costes de licencias, y se puede modificar para conseguir los resultados realmente necesitados.
- Está escrito en Python. Por lo que la robustez, modularidad y facilidad de mantenimiento del código está garantizada.
- Usa la base de datos libre MySQL.
- Usa el sistema R de estadística y generación de gráficas, por lo que las gráficas resultantes son de una calidad y utilidad desconocida en los sistemas SCADA.
- Funciona en sistemas con arquitectura de PC, baratos y universales.
- Funciona en sistemas basados en Linux. Libre y de gran estabilidad.

1.2. Qué es un SCADA

Un Scada es un sistema de Supervisión, Control y Adquisición de Datos, y se usa principalmente en entornos industriales, [ver definición de Scada](#).

1.3. Más concretamente: Qué hace Likindoy.

Likindoy puede hacer infinidad de cosas, pues tiene una gran flexibilidad y las posibilidades y combinaciones son ilimitadas, pongamos algunos ejemplos:

1.3.1. Ejemplo: Gráficas diarias de proceso.

Al día de hoy Axaragua usa Likindoy en sus plantas de tratamiento de agua potable y de depuración de aguas residuales para generar gráficas diarias de la siguiente manera:

En cada planta existe un PC con linux que corre Likindoy-RTU: éste toma datos de sus PLC de planta (Momentum, de Schneider) cada 5 segundos, los almacena en un archivo de texto plano con un formato estándar y luego los comprime con bzip.

Un servidor central (también linux) corre Likindoy-HTR: cada hora, por ftp, el servidor central requiere los archivos de texto a las Likindoy-RTU y los carga a una base de datos mysql.

Usando R, a partir de los datos almacenados en la base de datos MySQL, se generan gráficas muy detalladas.

Dichas gráficas se envían por correo electrónico a los responsables de la planta antes del comienzo de cada jornada.

1.3.2. Ejemplo: Visualización de datos en tiempo real.

Otra utilidad que se encuentra en funcionamiento actualmente es:

En cada planta Likindoy-RTU toma datos también de un módulo de adquisición de datos ADAM 5000 (con Modbus TCP/IP), conectado a una estación meteorológica. (Velocidad y dirección del viento, temperatura, pluviometría...)

Likindoy-RTU guarda dichos datos en una base de datos SQLite, y los manda cada XX segundos vía UDP a XXX.

Estos datos, además, son almacenados en un archivo de texto plano y procesados para generar gráficas diarias como en el ejemplo anterior.

En XXX, se ha diseñado una página web en Ajax que representa los datos en tiempo real sobre un mapa de Google Maps.

Ved el resultado en <http://www.tiempo.axaragua.com>.

1.3.3. Otras aplicaciones.

El módulo Likindoy-HTR es capaz de tomar datos de muchas fuentes distintas, no sólo de equipos por Modbus TCP/IP: de páginas web con información diaria, de sus propios PC donde está funcionando, y de autómatas remotos marca Pesyr.

Tomando datos de la web se generan [gráficas diarias de la situación de los embalses](#).

Tomando datos de los propios PC obtenemos gráficas de tráfico de redes.

Los autómatas remotos marca Pesyr se realizan en Málaga y son baratos y muy robustos y se comunican por SMS, GSM y GPRS. Un ejemplo de gráfica tomada de estos autómatas es la Figura 1.

1.4. ¿Para qué se crea Likindoy?

Los objetivos que nos han llevado a crear Likindoy son varios:

1.4.1. Gráficas de proceso de gran calidad.

La calidad, la personalización y la utilidad de las gráficas que se ha conseguido son una de las razones de ser de Likindoy y una de las características que lo hacen distinto a cualquier otro SCADA. Para ilustrar ésto veamos un ejemplo. La Figura 1 (en la página siguiente), representa los datos registrados en una estación de bombeo de aguas residuales sencilla. Del análisis de la gráfica se puede apreciar que el caudal de la única bomba que está en funcionamiento es de 78 l/s, que el nivel sube muy lentamente, pues el aporte de agua es pequeño, que la bomba tarda poco en evacuar el agua que entra, que no ha habido cortes de tensión,

y sí una entrada de inspección (puertas abiertas) a las 15:30. En definitiva se consigue un chequeo completo del correcto funcionamiento de la instalación de un solo vistazo.

1.4.2. SCADA libre.

En la actualidad la mayoría de los procesos industriales están controladas por un extenso abanico de software propietario, cuyas desventajas son: el coste de las licencias, su obsolescencia, una gran dependencia tecnológica (no puedo realizar ciertas cosas porque el sistema no me deja), y, lo que es más importante, son sistemas «cerrados» poco configurables y adaptables al proceso real (necesitamos una solución determinada que el sistema no da), por lo que terminamos renunciando a conseguir ciertos resultados, o incorporando muchas herramientas distintas para acometer tareas sencillas.

Likindoy, que es software libre, no tiene coste de licencias, y tampoco es fácil que quede obsoleto, pues el usuario siempre puede ir mejorando y ampliando sus capacidades, a lo cual favorece mucho que esté desarrollado en Python, un lenguaje interpretado de alto nivel que resulta sencillísimo de leer y mantener.

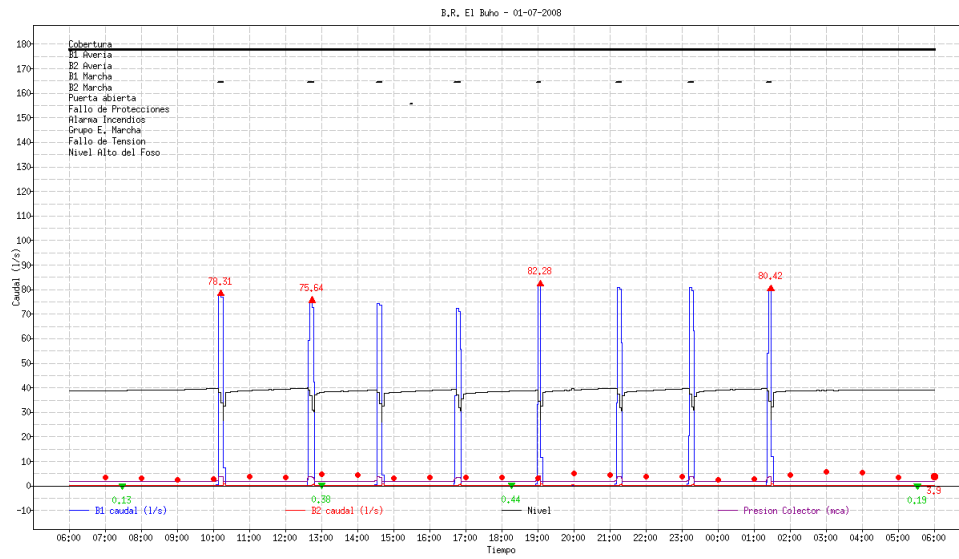


Figura 1: Ejemplo de gráfica generada por Likindoy-HTRs.

1.4.3. SCADA en PC, no en PLC.

La mayoría de los SCADAS industriales se basan en sistemas «no estándar», distintos para cada marca: distintas redes, protocolos, PLC propietarios, sistemas HMI distintos, etc.

Detrás de Likindoy reside el concepto de que para los sistemas de control modernos es necesario aplicar tecnologías que en la informática común se entiendan como estándar: ethernet para comunicación física, modbus TCP/IP para los protocolos, ftp para la transferencia de ficheros, MySQL para las bases de datos, Flash para el HMI, etc.,. De esta manera cualquiera que tenga conocimientos suficientes de informática puede montar y mantener un sistema de control complejo.

Por ello es una herramienta ideal para el técnico en informática que quiera adentrarse en el control industrial, y para el técnico de control industrial que o bien tenga una buena base informática, o cuente con apoyo de algún tipo en ese área.

1.5. Quién ha desarrollado Likindoy.

El desarrollador de Likindoy es el hacker Juan Miguel Taboada Godoy, quien ha empleado más de 2000 horas en la escritura del código, búsqueda de bugs, puesta en marcha.. etc...

Las especificaciones de Likindoy son de Juan José Denis

Corrales, Ingeniero Industrial del ICAI y gerente de Axaragua, habiendo participado en las tareas de análisis de objetivos, evaluación de alternativas, búsqueda de fallos, y parte de la documentación.

Likindoy es una iniciativa de Axaragua, quien ha aportado la financiación del proyecto ayudado por la Oficina de Innovación de la Junta de Andalucía, y de Juan Miguel Taboada Godoy, que ha aportado grandes cantidades de esfuerzo personal extraordinario. Recientemente se ha unido el hacker Michael Griffin quien ha realizado importantes mejoras en los protocolos Modbus sobre TCP/IP.

1.6. Sistema de desarrollo.

La creación de Likindoy ha seguido el modelo de [desarrollo iterativo y creciente](#), y pretendemos usar más técnicas de [programación extrema](#), se consiguieron los primeros resultados prácticos en las primeras semanas, desde entonces ha habido una interacción constante con el usuario y por ende una retroalimentación continua. Ha sido completamente reescrito 2 veces. El módulo Likindoy RTU se está reescribiendo en la actualidad.

1.7. Evolución temporal.

enero - junio de 2005 El proyecto se inició a finales de 2004. Buscando un sistema funcional que generara gráficas

de manera inmediata, el diseño se basó en programación en R, que tomaba las decisiones a la hora de generar cada gráfica. Vimos que la calidad de las gráficas era extraordinaria y que merecía la pena seguir avanzando.

julio de 2005 - junio de 2006 Durante finales del año 2005 y toda la primera mitad del año 2006 el proyecto se reescribió en Python buscando extender las funcionalidades. Cuando tuvimos un sistema que cubría nuestras necesidades iniciales vimos que dividiendo el sistema en módulos, podríamos crear un sistema SCADA completo.

julio de 2006 - octubre de 2007 Durante los meses de Julio y Agosto del año 2006 (julio y agosto) el proyecto fue reescrito completamente, se crearon clases para casi todo, se definieron las configuraciones por defecto, etc...

Ayudados por la subvención de la Consejería de Innovación Tecnológica, se impulsaron las siguientes funcionalidades:

- Aceleramos la adquisición de datos.
- La modularización del sistema, de forma que se permitiesen datos de entrada y formatos de salida distintos a los inicialmente considerados.
- Programación de una nueva fuente de datos: los Momentum de Schneider. Esta modificación supuso tam-

bién la creación de unas “Unidades Remotas de Telecontrol”, que son ordenadores personales (PC) que se comunican con los autómatas y envían los datos registrados en archivos de texto a la unidad central. Esta modificación también ha supuesto la programación del protocolo de comunicación Modbus sobre TCP-IP, que es libre y se está popularizando entre los fabricantes de hardware.

- Envío de los datos adquiridos por dichas remotas, por UDP u otros protocolos, a un servidor web, de forma que se pueda ofrecer información relevante al ciudadano en tiempo real (temperaturas, intensidad y dirección del viento...). puede ser visto en <http://tiempo.axaragua.com>.
- Documentación del proyecto.

octubre de 2007-enero de 2008. En este período nos centramos especialmente en el diseño de una RTU programada en C que funcionara directamente dentro de los Advantech ADAM 5510 que tenían una CPU 80188. El proyecto avanzó muchísimo y cuando estaba al 85 % aproximadamente descubrimos que el hardware no era lo suficiente estable para lo que buscábamos (se bloqueaba sin dar posibilidad a controlarlo remotamente para reiniciarlo u otro) a esto le añadimos que en cuanto el sistema creció bastante el programa era además demasiado lento, echábamos de menos

la multitarea y finalmente un mes y medio más tarde el proyecto quedó abandonado.

Su código aún no se ha liberado, pero lo haremos próximamente.

En la actualidad. En la actualidad, además del mantenimiento y mejora continuas del software en funcionamiento, estamos trabajando en varias áreas:

Reescribiendo Likindoy-RTU, trasladando nuestra experiencia con la RTU en C, con ello pretendemos conseguir una RTU totalmente versátil, que además de adquisición de datos y servidor de éstos vía UDP, sea capaz de envío de alarmas, toma de decisiones y actuación en campo. Es decir, todas las capacidades de una RTU comercial. (PLC)

A la vez ampliamos Likindoy-RTU para su comunicación con unos PLC de la marca Pesyr, por GPRS. es un proyecto paralelo en el que buscamos realizar una versión del servidor UDP de Likindoy que además soporte la entrada de datos de unidades remotas de PESYR que enviarán sus capturas en paquetes UDP a través de GPRS al servidor UDP de Likindoy.

En el futuro. En el futuro pretendemos preparar un sistema HMI, es decir, la visualización de los datos en sinópticos y de alarmas en tiempo real, para ello pretendemos usar Flash.

2. Instalación del sistema

2.1. Requisitos

El sistema mínimo que likindoy necesitará para funcionar es Python, mediante este se le irá informando del resto de dependencias y carencias del sistema gracias a su instalador integrado.

No obstante si desea conocer el resto de requisitos a gran escala anote:

1. Un servidor MySQL accesible desde la máquina (local o remoto) y librerías para Python para acceder a MySQL. ¹
2. Sistema R-CRAN ² y librería rpy. ³
3. Librería GnuPGInterace para Python. ⁴
4. Librería Image para Python. ⁵
5. Librería SQLite para Python. ⁶

¹Almacena los datos recogidos de todas las señales.

²Lenguaje de análisis estadístico R ([ver proyecto](#))

³Dispone el entorno necesario para generar las gráficas.

⁴Obligado por la librería de encriptación, permite al servidor UDP usar encriptación PGP además de la ya incluida TripleDES.

⁵Genera ciertas marcas especiales en las gráficas.

⁶Permite al sistema RTU almacenar localmente la información que va recibiendo del sistema así como su estado interno.

6. Librería Paramiko para Python. ⁷

7. Librería XML para Python. ⁸

2.2. Instalación automática

Acuda al directorio doc/ en el que podrá encontrar el programa **install.py** que se encargará de comprobar los paquetes que faltan y de crear el usuario y base de datos correspondiente para usar Likindoy. Al finalizar la ejecución del instalador correctamente, acuda al punto 2.4 de este documento.

2.3. Instalación manual

Para realizar la instalación manual en un sistema basado en Debian deberá asegurarse de tener instalado los siguientes paquetes antes de continuar:

1. **python-gnupginterface:** Interfaz Python a GnuPG (GPG)
2. **python-mysqldb:** Interfaz Python a MySQL
3. **python-paramiko:** Interfaz Python para gestión de conexiones SSH2

⁷Permite descargar ficheros de datos mediante SFTP.

⁸Necesaria para leer los ficheros de OpenOffice Calc

4. **python-rpy:** Interfaz Python al entorno y lenguaje GNU R

5. **python-sqlite:** Interfaz Python a SQLite 2

Nota: Tenga en cuenta que el proyecto está en continuo desarrollo y ha sido comprobado bajo distribuciones basadas en Debian. Cualquier tipo de ayuda será bienvenida así como el documento de instalación para otro tipo de distribuciones o incluso otros sistemas operativos.

A continuación describiremos una configuración mínima para empezar a trabajar. Para una descripción completa del sistema y la instalación atienda al Manual de Usuario Avanzado en doc/html/.

a) Creación de la Base de Datos y usuario de acceso con la siguiente configuración:

- **Servidor:** Localhost (127.0.0.1)
- **Puerto:** 3306
- **Usuario:** likindoy
- **Clave:** likindoyclave
- **Base de datos:** likindoy_test

b) Creación de las siguientes tablas:

```
CREATE TABLE `datos` (
  `id` int(11) NOT NULL,
  `server` timestamp NOT NULL default CURRENT_TIMESTAMP,
  `fecha` date NOT NULL,
  `hora` time NOT NULL,
  `valord` tinyint(1) default NULL,
  `valora` float default NULL,
  PRIMARY KEY (`id`,`fecha`,`hora`)
);
CREATE TABLE `ids` (
  `id` int(11) NOT NULL auto_increment COMMENT 'ID consecutivo',
  `md5` varchar(32) NOT NULL COMMENT 'ID MD5',
  PRIMARY KEY (`id`),
  UNIQUE KEY `md5` (`md5`)
);
```

2.4. Cargando los primeros datos

Antes de ejecutar cualquier comando vamos a renovar la lista de tareas por hacer para así asegurarnos que Likindoy acepta nuestras peticiones pensando que todo el trabajo está aun por hacer.

```
> # -----
> # --- Si en este paso se produce algun error en la familia se eliminan
> # --- significa que existe algun problema en la familia se eliminan
```

```
> # -----
> ./renovar.py
```

Preparamos los datos a cargar en la base de datos, para ello vamos al directorio raíz de Likindoy y ejecutamos el siguiente comando (con ello conseguimos disponer en el directorio `data/reg/` (directorio de registros) un conjunto de ficheros de registros que se han descargado anteriormente):

```
> cp data/reg.bak/* data/reg/
```

Cargamos los datos a la base de datos

```
> ./cargar.py
```

2.5. Generando las primeras gráficas

Creamos las gráficas que hay en los ejemplos:

```
> ./graficar.py
```

Podemos encontrar las gráficas generadas en los siguientes directorios:

- `/data/dib/` : Para uso normal (generalmente se manda por email y se eliminan)

- **./data/dib.bak/** : Copia de seguridad
- **./src/tmp/dib/hist/** : Para historicos (se procesan en una web u otro programa)

Para volver a pintar es necesario ejecutar el comando antes de crear las gráficas:

```
> ./renovar.py
```

2.6. Registrando los primeros datos

También podemos registrar nuestros primeros datos usando el módulo web, para ello ejecuta el siguiente comando:

```
> # -----
> # --- En este punto el programa puede indicar que no es ---
> # --- necesario sincronizar, lo que significa que la ---
> # --- descargado los ficheros de hoy. Si quieres que la ---
> # --- descarga sea mas asidua consulta el fichero ---
> # --- ./src/etcbin/adquirir_web_default.py), operador y administrador (entes activos).
> # --- y busca la linea: ---
> # --- config.download_tempo(60*60*24) ---
> # -----
> ./adquirir_web.py
```

Nota: Al declarar los registradores NET_UDP en la configuración debería insertarse en la declaración la llave 3DES que se desee. Por defecto en la configuración no se ha incluido ninguna llave, como se muestra en el siguiente ejemplo:

```
net_udpPLC1_local=NET_UDP("127.0.0.1","plc1",,
```

aunque esto en el sistema equivale a:

```
net_udpPLC1_local=NET_UDP("127.0.0.1","plc1",,
```

pero debería cambiarse a algo como:

```
net_udpPLC1_local=NET_UDP("127.0.0.1","plc1",,
```

donde "123456789012345678901234" sea la nueva para codificar su comunicación UDP.

3. Manuales

Existen 3 manuales acorde a la persona: usuario (ente pasivo), operador y administrador (entes activos).

3.1. Manual de usuario

El usuario de Likindoy es únicamente recibe los resultados de este. Esto equivale a recibir por email las gráficas

y entenderlas en la medida en que el usuario y el operador acordaron en lo referente a la simbología, estilo y colores de las líneas y de los datos representados en las gráficas.

Igualmente para los usuarios de los sistemas Scada en tiempo real el resultado viene dado directamente por las decisiones tomadas en el momento de generar la configuración del sistema y por lo tanto el operador será el encargado de indicar la simbología y significado de cada elemento del sistema.

A modo de ejemplo, los usuarios de Likindoy podrían ser los operarios de planta que reciben las gráficas todas las mañanas, y que gracias a estas pueden salir temprano a resolver las incidencias o realizar los mantenimientos preventivos que mediante los datos contenidos en las gráficas se decidan. También podrían ser los usuarios de la web que consultan el estado de las estaciones meteorológicas en la costa de Málaga.

3.2. Manual de operador

El operador es la persona que se limita únicamente a modificar los ficheros de configuración dispuestos en el directorio etc/, conoce la estructura general del sistema y por lo tanto mediante esta se encarga de definir cada uno de los parámetros.

Para el operador se dispone a continuación un manual de uso que le servirá de referencia inicial para comen-

zar a trabajar con Likindoy. Posteriormente el operador deberá hacer referencia a la documentación disponible en `doc/html/index.html` para conocer las opciones avanzadas contenidas dentro de cada uno de las funciones disponibles.

3.2.1. Preguntas antes de comenzar

Con estas sencillas preguntas tratamos de contextualizar al operador dentro del sistema y hacerle entender el sentido de su trabajo frente a Likindoy y las razones de por qué se le ha limitado el acceso al directorio etc/ únicamente.

¿Qué es lo que tengo en el directorio etc/ El directorio etc/ contiene toda la configuración *específica* del sistema, esto es:

- **Telemandos:** declaración de todos los telamandos del sistema indicando a Likindoy donde debe recoger la información, claves de acceso, etc... Cada fichero de recogida se ha separado por el modo en que los datos son recogidos, de tal modo que hemos nombrado a todos como `adquirir_*.py` y cada cual corresponde a su propio módulo dentro de Likindoy. Por ejemplo: `adquirir_signals.py` contiene la configuración de todos los telemandos `Likindoy-compatibles`, o bien el fichero `adquirir_pesyr.py` contiene la configuración de todos los telemandos `PESYR`, o el `adquirir_ftp.py`

para que podamos bajarnos los datos de ciertos telemandos directamente mediante FTP.

- **cargador.py:** contiene únicamente el orden en que deben ser cargados los ficheros, de tal modo que así podamos disponer de un conjunto de datos antes que otro aunque este primero haya llegado después. Es muy útil para acelerar la generación de gráficas específicas.
- **emails.py:** contiene información específica sobre cada usuario que recibirá las gráficas generadas.
- **senales.ods:** es un fichero OpenDocument Spreadsheet (OpenOffice Calc) y contiene toda la información relativa a las señales del sistema. Indica desde su nombre interno, descripción, telemando al que pertenece, si es analógica o digital, en que conector del telemando se encuentra, como debe ser procesada en la adquisición, etc...
- **Otros:** existen otros ficheros relativos a configuraciones de subprogramas de Likindoy, desde gestión de pings para estadísticas de red, servidores UDP, gestión de alertas, etc...

¿Por qué puedo tocar sólomente en este directorio? Es una buena pregunta, debido a que existe un segundo directorio de configuración `supergrupo` de `etc/` y se encuentra

en `src/etcbin/`. No se permite al operador tocar en este directorio, pues en él se localiza la información genérica del sistema, incluyéndose en él detalles de acceso a servidores MySQL, llaves DSA para uso de SSH, sistema de gestión de señales, etc...

Casi todas las opciones definidas por defecto en este directorio puede ser sobreescritas mediante las configuraciones disponibles en `etc/` por lo que un operador no debería tener necesidad de tocar en este directorio. El resto de opciones que no son transparentes en `etc/` se refieren a configuraciones que sólo el administrador debe controlar y que requieren un conocimiento más avanzado del sistema.

¿Cómo debo orientarme dentro de Likindoy? Es muy sencillo, comienza por trabajar de un modo convergente, es decir, enfoca tus esfuerzos en orden:

1. Configura la adquisición de datos
2. **Prueba que adquieres datos correctamente**
3. Configura el orden del sistema de carga
4. Añade todas las señales que necesites
5. **Prueba que el sistema de carga almacena la información en la base de datos**
6. Configura todas las gráficas

7. **Prueba que el sistema de gráficas genera los ficheros correctamente**
8. Configura el sistema de envío de emails
9. **Prueba que el sistema de emails envía las gráficas correctamente**
10. Comienza a trabajar con los sistemas adyacentes: estadísticas de red, rtu, servidores UDP, etc...

Mayúsculas y minúsculas Asegúrese de usar mayúsculas y minúsculas adecuadamente en todo momento ya que el sistema es sensible a estas y considerará cadenas distintas “Trapiche”, “tRAPICHE” y “TRAPiChe”.

3.2.2. La adquisición de datos

```
# Ejemplo
t=TELEMANDO("Trapiche")
t.config(config,"192.168.x.y","netdata")
t.user("username")
t.passwd("my_secret_password")
listado_telemandos.append(t)
```

La información que requiere un telemando es poca:

1. Primero declaramos el telemando

2. Indicamos al telemando la configuración que queremos usar: dirección IP del telemando y el nombre del fichero de datos que nos queremos traer del mismo.
3. Indicamos el nuevo usuario (para que no use el usuario por defecto). Este campo es opcional.
4. Indicamos la nueva clave (para que no use la clave por defecto). Este campo es opcional.
5. Añade el telemando a la lista de telemandos que será procesada en el script principal de Likindoy

3.2.3. La sencillez del cargador

Del cargador hay muy poco que decir debido a que casi todo se hace automáticamente. Cuando se ejecuta, busca los ficheros a cargar en el directorio `data/reg/` y cuando termina de procesar el fichero lo borra de dicho directorio. La ejecución es selectiva en función de los ficheros que se encuentre en el directorio de modo que primero se cargan los ficheros indicados en la configuración del cargador y después el resto de ficheros por orden alfabético. Después de cada lazo del bucle se comprueba de nuevo la lista de ficheros para asegurarnos que siempre cargamos los más prioritarios.

El orden de ejecución de los ficheros se define dando el nombre de los telemandos en el orden en que deseamos sean cargados sus ficheros:

```
sql.add_ordered("Trapiche")
sql.add_ordered("Segundo")
sql.add_ordered("Tercer")
...
```

3.2.4. Los misterios del senales.ods

El fichero de señales es el más importante de todo Likindoy debido a que interconecta al “mundo real” con el “mundo virtual”, de tal modo que prácticamente refleja toda la base de conocimientos del sistema.

La tabla de datos de senales.ods contiene los siguientes campos:

- **Descripción:** de la señal, no tiene ningún uso específico en Likindoy salvo aclarar la fila en la hoja de cálculo.
- **A/D:** indica si la señal es analógica (a) o digital (d).
- **Frecuencia:** Es la frecuencia de recogida de la señal, de tal modo que el sistema interpretará que la comprensión de los datos puede venir hasta con una frecuencia máxima de la indicada aquí, si pasan más segundos, minutos, horas o días de los indicados en este campo y el fichero de datos no contiene ninguna autoreferencia a la frecuencia entonces el cargador automáticamente indicará que faltan datos en el rango implicado. Esto no afecta a la carga aunque sí al generar las gráficas.

- **En BD:** indica si esta señal deberá almacenarse en la Base de Datos. Es especialmente útil para señales que deseamos tener en el senales.ods para ser procesadas por otro subsistema, pero no las queremos almacenar en la base de datos.
- **RTU:** es el nombre del telemando al que pertenece dicha señal.
- **ID:** es un identificador único para cada señal. Si usted repite el identificador en 2 señales, todos los sistemas le avisarán de ello: cargador, generador de gráficas, sistema de envío de emails, gestor de RTU, servidor UDP y cualquier módulo que use el fichero de senales.ods mediante la librería de señales.
- **Dirección, zona y variante:** el primero contiene la dirección física de la señal, mientras que el segundo contiene la zona de la que obtener los datos dentro de esa dirección. Todas las señales **reales** tienen una dirección física de acceso (sólo las señales virtuales generadas por realimentación no tienen una dirección física, lo cual es lógico), dependiendo del tipo de señal la dirección tendrá un formato u otro, también cada tipo de dirección tiene su subconjunto de zonas. De tal modo que:
 - **Schneider - Momentum:** usa un formato numérico compatible con la dirección propia asignada

por dichos sistemas. Un ejemplo de dirección es 400413 para una señal que contiene un valor real o 100043 para una señal digital.

- Advantech - ADAM 5000 TCP: usa un sistema de direccionamiento basado en slot y posición dentro del slot. Un ejemplo de dirección es 2,0 que se refiere al slot 2 primer conector (conector 0). Este tipo de direcciones permiten 2 tipos zonas “ra” (lectura analógica *) y “rd” (lectura digital *).
- Netstatistic: para el sistema de análisis de red la dirección puede ser el dispositivo sobre el cual se desea analizar la información o la dirección ip sobre la que se quiere operar, cada cual necesitará además indicar la zona desde la que se van a extraer los datos. Un ejemplo sería 192.168.0.23, también eth0 o incluso www.centrologic.com. Este tipo de direcciones permite varios tipos de zonas:
 - Para dispositivos de soporta:
 - ◇ **bi**: Bandwith Input o ancho de banda de entrada. Representa el uso de la banda de

entrada. *=que se aplica en la columna variante

- ◇ **bo**: Bandwith Output o ancho de banda de salida. Representa el uso de la banda de salida. *=que se aplica en la columna variante
- ◇ **pi**: Packages Output o número de paquetes de entrada. Representa el uso de la banda de entrada. *=que se aplica en la columna variante
- ◇ **po**: Packages Output o número de paquetes de salida. Representa el uso de la banda de salida. *=que se aplica en la columna variante
- Para direcciones soporta:
 - ◇ **hp**: Hay ping *
 - ◇ **pe**: Ping enviados *
 - ◇ **pr**: Ping recibidos *
 - ◇ **pp**: Ping perdidos (%) *
 - ◇ **pm**: Ping mínimo *
 - ◇ **pa**: Ping medio (average) *
 - ◇ **px**: Ping máximo *
 - ◇ **pv**: Ping dispositivo *

Además todas las zonas que sean Likindoy-Compatible (señaladas con *) soportan además

de su nombre tal cual (que contiene el valor instantáneo) añadir el sufijo “med” (si se desea el valor medio entre registros), el sufijo “max” (si se desea el valor máximo), el sufijo “min” (si se desea el valor mínimo) o el sufijo “var” (si se desea el valor de la varianza entre registros).

- **Web:** para el módulo de adquisición web depende de los datos declarados, lo cual va directamente declarado en la configuración de dichos “telemandos”. Un ejemplo para datos obtenidos de la página web del Instituto Nacional de Meteorología podrían ser “VientoFuerzaTarde” o de datos extraídos de la Agencia Andaluza del Agua podría ser “ConcepcionCap”. La zona para esta dirección es siempre “data”. *
- **PESYR - Infocel:** para los datos extraídos de sistemas Infocel de Pesyr, donde se usa un direccionamiento definido por ellos basado en letras y números. Un ejemplo sería “EA0” para datos extraídos de la entrada analógica 0. *=que se aplica en la columna variante (no soporta la variante “var” de varianza).
- **Interprete RTU:** algunas señales necesitan ser interpretadas después al obtenerse del fichero y antes de ser cargadas a la base de datos, para eso se usa este campo en el que se indica la función de interpretación de

datos que se debe usar. No es una función de ajuste de rangos, sino una función para interpretarlos.

- **Lógico inicio, lógico fin, físico inicio y físico fin:** se usan para hacer un reajuste de los rangos de la señal, de tal modo que en lógico inicio/fin se define el valor lógico que la señal deberá tener después de ser modificado el rango y el valor físico inicio/fin corresponde al valor físico que se está obteniendo del sistema. Si no se pone nada se usará 0 para valor de inicio y 65535 para valor de fin.

Todos estos campos serán detallados más profundamente en documentos anexos específicos de cada sistema.

3.2.5. Introducción a la generación de gráficas

En este apartado se introducirán las bases para la generación de gráficas, pero para aspectos avanzados de la generación de gráficas se recomienda visitar el directorio `doc/html/index.html` que contiene la documentación avanzada del uso de “addline”.

Un ejemplo de gráfica podría ser:

```
g=GRAFICA("El Buho",s)
g.config(config)
g.sizey(-10,180,10,5)
g.addaxis(0,"izq","Caudal (l/s)")
g.addaxis(0,"inf","Tiempo")
```

```

g.sizeX("date:ayer + 6h","date:hoy + 6h","horas")evento_valor)

g.addline("Fallo de Tension","ElBuho_FalloTen
evento_valor)
g.addline("Cobertura","ElBuho_Cobertura","a",evento_valor,color_1,color_2,color_3,
cobertura_valor)

g.addline("Fallo de Tension","ElBuho_FalloTen
evento_valor)
g.addline("Cobertura","ElBuho_Cobertura","a",evento_valor,color_1,color_2,color_3,
cobertura_valor)

g.addline("B1 Averia","ElBuho_AveriaB1","d",evento_valor,color_1,color_2,evento_valor)
g.addline("B2 Averia","ElBuho_AveriaB2","d",evento_valor,color_1,color_2,evento_valor)
g.addline("B1 Marcha","ElBuho_MarchaB1","d",evento_valor,color_1,color_2,evento_valor)
g.addline("B2 Marcha","ElBuho_MarchaB2","d",evento_valor,color_1,color_2,evento_valor)
g.addline("B1 caudal (l/s)","ElBuho_CaudalB1",
evento_valor,color_1,color_2,evento_valor)
g.addline("B2 caudal (l/s)","ElBuho_CaudalB2",
evento_valor,color_1,color_2,evento_valor)
g.addline("Nivel","ElBuho_Nivel","a",escalera,
evento_valor,color_1,color_2,evento_valor)
g.addline("Puerta abierta","ElBuho_PuertaAbierta1","d",evento_valor,color_1,color_2,
evento_valor)
g.addline("Presion Colector (mca)","ElBuho_PresionColector",
evento_valor,color_1,color_2,evento_valor)
g.addline("Fallo de Protecciones","ElBuho_FalloProtecciones","d",evento_valor,color_1,color_2,
evento_valor)
g.addline("Caudal Medio 24h","( ElBuho_CaudalMedio24h",
evento_valor,color_1,color_2,evento_valor)
g.addline("Alarma Incendios","ElBuho_DetectorHumo","hoy_06:00:00,autoescalar",
evento_valor,color_1,color_2,evento_valor)
g.addline("Grupo E. Marcha","ElBuho_MarchaGrupo","d",evento_valor,color_1,color_2,
evento_valor)
g.addline("Caudal Medio 24h2","( ElBuho_CaudalMedio24h2",
evento_valor,color_1,color_2,evento_valor)
g.addline("Puerta abierta 2","ElBuho_PuertaAbierta2","d",evento_valor,color_1,color_2,
evento_valor)

```

```
g.addline("Caudal Total Maximo (l/s)", "( ElBuho_CaudalB2 notNeg ) max:4", "a", "max", rojo, puntos_medianos, minimos)
```

Declaramos el tamaño del eje y como va a crecer la regilla, indicamos que irá de 10 a 180, poniendo líneas horizontales cada 10 unidades y poniendo una marca cada 5 unidades:

```
g.addline("Caudal Total Minimo (l/s)", "( ElBuho_CaudalB1 notNeg ) + ( ElBuho_CaudalB2 notNeg ) min:4", "a", "min", verde, puntos_medianos, minimos)
```

```
g.leyenda("B1 caudal (l/s)")
g.leyenda("B2 caudal (l/s)")
g.leyenda("Nivel")
g.leyenda("Presion Colector (mca)")
```

```
g.outpng("68 BR El Buho - %iayer.png", "B.R. El Buho")
graficas.append(g)
```

Añade el texto del eje izquierdo como "Caudal (l/s)":

```
g.addaxis(0, "izq", "Caudal (l/s)")
```

En primer lugar se genera el la gráfica entregando su nombre "El Buho" y el objeto que contiene toda la información de las señales "s":

```
g=GRAFICA("El Buho", s)
```

Cargamos la configuración por defecto a la gráfica:

```
g.config(config)
```

Declaramos el tamaño del eje X, indicamos que irá desde ayer a las 6:00am hasta hoy a las 6:00am y que el corte del eje x se hará por horas:

```
g.sizeX("date:ayer + 6h", "date:hoy + 6h", "horas")
```

Añade el texto del eje inferior como "Tiempo":

```
g.addaxis(0, "inf", "Tiempo")
```

"addline" añade líneas a la gráfica. El orden en que se añaden las líneas afectará al orden en que son pintadas de tal modo que la última añadida quedará por encima de las demás. La correspondencia de cada campo es:

1. Nombre de la línea

2. Ecuación que se mostrará
3. Indica si se usará una representación analógica (a) o digital (d)
4. Representación a usar para la línea o equivalente
5. Color principal
6. Grosor de la línea
7. Tipo de línea a usar o equivalente
8. Existen otros campos que extienden el uso y se puede consultar en la documentación avanzada.

Un ejemplo claro de señales representadas en horizontal (digitales y eventos):

```
g.addline("Cobertura", "ElBuho_Cobertura", "a", evento, cobertura_color, 3, cobertura_valor)
g.addline("B1 Averia", "ElBuho_AveriaB1", "d", evento, evento_color1, 2, evento_valor)
g.addline("B2 Averia", "ElBuho_AveriaB2", "d", evento, evento_color1, 2, evento_valor)
g.addline("B1 Marcha", "ElBuho_MarchaB1", "d", evento, evento_color1, 2, evento_valor)
g.addline("B2 Marcha", "ElBuho_MarchaB2", "d", evento, evento_color1, 2, evento_valor)
g.addline("Puerta abierta", "ElBuho_PuertaAbierta", "a", escalera, nivel, "ElBuho_Nivel", "a", escalera,
```

```
evento_valor)
```

```
g.addline("Fallo de Protecciones", "ElBuho_FalloProtecciones", "a", evento_color1, 2, evento_valor)
```

```
g.addline("Alarma Incendios", "ElBuho_DetectorAlarma", "a", evento_valor)
```

```
g.addline("Grupo E. Marcha", "ElBuho_MarchaGrupo", "a", evento_valor)
```

```
g.addline("Puerta abierta 2", "ElBuho_PuertaAbierta2", "a", evento_valor)
```

```
g.addline("Fallo de Tension", "ElBuho_FalloTension", "a", evento_valor)
```

```
g.addline("Nivel Alto del Foso", "ElBuho_NivelAlto", "a", evento_valor)
```

Un ejemplo claro de señales analógicas sería:

```
g.addline("B1 caudal", "ElBuho_CaudalB1", "a", evento_color1, 2, evento_valor)
```

```
g.addline("B2 caudal", "ElBuho_CaudalB2", "a", evento_color1, 2, evento_valor)
```

```
g.addline("Presion Colector (mca)", "ElBuho_Presion * 10", "a", escalera,
color_presion, 1, solida)
```

Un ejemplo del uso de ecuaciones más complejas sería:

```
g.addline("Caudal Medio 24h", "( ElBuho_CaudalB1: ((date:hoy_06:00:00-24d, da
te:hoy_06:00:00,autoescalar) notNeg ) + ( ElBuho_CaudalB2: ((date:hoy_06:00
:00-24d,date:hoy_06:00:00,autoescalar) notNeg )
ntos_medianos,medias)
```

```
g.addline("Caudal Medio 24h2", "( ElBuho_CaudalB1 notNeg ) + ( ElBuho_Caud
alB2 notNeg ) med:1", "a", "med", rojo, puntos_grandes,mediasUnica)
```

```
g.addline("Caudal Total Maximo (l/s)", "( ElBuho_CaudalB1 notNeg ) + ( ElB
uho_CaudalB2 notNeg ) max:4", "a", "max", rojo, puntos_medianos,maximos)
```

```
g.addline("Caudal Total Minimo (l/s)", "( ElBuho_CaudalB1 notNeg ) + ( ElB
uho_CaudalB2 notNeg ) min:4", "a", "min", verde, puntos_medianos,minimos)
```

Indica los nombres de las líneas que van a aparecer en las leyendas, las leyendas aparecerán en el orden en que sus líneas han sido declaradas:

```
g.leyenda("B1 caudal (l/s)")
g.leyenda("B2 caudal (l/s)")
g.leyenda("Nivel")
```

```
g.leyenda("Presion Colector (mca)")
```

Declara la información de salida de la gráfica, en este caso se indica el nombre del fichero (donde %iayer se traduce como la fecha inversa de ayer año-mes-día), el título de la gráfica (donde %ayer se traduce como la fecha en orden normal día-mes-año), y los dos siguientes campos corresponden a la resolución horizontal y vertical.

```
g.outpng("68 BR El Buho - %iayer.png", "B.R. E
```

Finalmente añadimos la gráfica al listado de gráficas que se procesarán en el script principal:

3.2.6. Obtener los resultados por email

El sistema de email ha sido desarrollado para que recoja todos los emails del directorio data/dib/ y los envíe a los usuarios correspondientes en función de su lista de gráficas, una vez terminado borra los emails del directorio data/dib/

Para configurar el fichero primero generamos el objeto AUTOEMAIL:

```
a=AUTOEMAIL("Emails Axaragua")
```

le pasamos la configuración por defecto y el listado de gráficas del fichero de configuración de graficas:

```
a.config(config, graficas)
```

configuramos el asunto del email definitivo y el asunto del email temporal (cuando no se mandan todas las gráficas):

```
a.subject("[LKD] Graficas del %semana %fecha" + texto. Centrologic.")
a.subject_temp("[LKD-TMP] Graficas del %semana %fecha" + texto. Centrologic.)
```

generamos un listado de nombres de gráficas así como el orden en que deseamos estas sean enviadas:

```
listadoGraficas=["Grafica Numero 1", "Grafica Numero 2", "Otra grafica"]
listadoGraficasExtendido=["Grafica X", "Grafica Y"] + listadoGraficas
```

añadimos a cada uno de los usuarios que van a recibir gráficas:

```
a.add("Nombre completo usuario interesado A", "Nombre corto usuario A", "direccion@email.uno, direccion@email.dos, direccion@email.tres", listadoGraficas)
```

```
a.add("Nombre completo usuario interesado B", "Nombre corto usuario B", "direccionB@email.uno, dirB@email.dos, dirB@email.tres", listadoGraficas)
```

3.2.7. La temida RTU

De la RTU casi no vamos a decir nada de momento debido a que estamos reimplementando gran parte de su núcleo y cambiando la filosofía de uso. No obstante hemos de indicar que Likindoy-RTU (la versión estable actual) dispone de Soporte para Advantech ADAM5000/TCP y Schneider Momentum. Su configuración es un tanto especial:

Una objeto RTU necesita los siguientes elementos:

- Nombre interno
- Un objeto REGISTRADORES
- Un objeto EVENTS (parámetro sólo disponible en la actualidad para el registrador de Advantech ADAM 5000 TCP)
- Listado de señales obtenidos del senales.ods

¿Qué es un registrador Un registrador es un elemento que se encarga de registrar o almacenar los datos de un modo concreto.

¿Qué tipo de registradores existen en Likindoy?

- **FILE SIGNALS:** registra los datos en un fichero de tipo SIGNALS. Este tipo de ficheros es propio de Likindoy y fue diseñado pensando en la claridad de los

datos y facilidad de uso. Es el tipo de fichero estándar para todos los datos registrados por RTUs de Likindoy.

- **NET_UDP:** al registrar los datos en un objeto de este tipo los datos son enviados a un servidor externo mediante paquetes UDP. Es el formato estándar usado por Likindoy para exportar información en tiempo real a otros sistemas.
- **Otros:** es posible realizar nuevos registradores con bastante facilidad implementando el interfaz REGISTRADOR que se puede encontrar en la librería INTERFACES.

Crear un objeto FILE_SIGNALS: El primer parámetro es el formato del fichero. El formato del fichero que se va a generar depende del modelo de registrador físico con el que se va a trabajar, eligiendo un tipo u otro de fichero modificamos también el grupo de funciones aceptadas por el mismo en el momento en el que se registran. Además el indicar el tipo de fichero añade información de control extra al fichero. Puede ser compatible con los siguientes modelos de RTUs:

- **advantech:** indica al registrador que se van a registrar datos de un sistema Advantech.
- **momentum:** indica al registrador que se van a registrar datos de un sistema Schneider Momentum.

- **netstatistics:** indica al registrador que se van a registrar datos de un sistema que está usando el módulo NETSTATISTICS de Likindoy.

- **web:** indica al registrador que se van a registrar datos de un sistema que está usando el módulo WEB de Likindoy.

El segundo parámetro es el nombre del fichero (sin la extensión “signals.dat”).

El tercer parámetro es “compress=True” para indicar al registrador que comprima el resultado (la extensión entonces será “signals.dat.gz”).

Este registrador registra todos los datos que le son entregados.

Un ejemplo sería:

```
registrador=FILE_SIGNALS("advantech", "tiempo",
```

Crear un objeto NET_UDP: Al crear un objeto NET_UDP se necesitan 5 parámetros:

- Dirección IP del sistema destino
- Identificador propio del objeto
- Listado de señales obtenidas del senales.ods
- Llave TripleDES para encriptación de los paquetes

- Puerto destino (por defecto 57221)

A continuación es necesario declarar cada una de las señales que se van a enviar al servidor remoto, es decir, el envío de datos es selectivo. Para ello y por comodidad podemos usar la función load del objeto, de tal modo que le indicamos con ella al registrador que envíe al servidor remoto todos los datos del telemando indicado, de tal modo que el registrador enviará los datos al servidor remoto si la señal que se trata de registrar en él está en el senales.ods registrada bajo el telemando indicado en el load.

Un ejemplo sería:

```
net_udpEstacion=NET_UDP("192.168.2.1", "tiempovelez", "abdefghijklmnopqrstuvwxyz")
net_udpEstacion.load("VelezTiempo")
```

El objeto REGISTRADORES: Este objeto requiere una lista de REGISTRADORES, esto es “[objeto_registrador_1,objeto_registrador_2,..]. El objeto registradores será usado por la RTU para enviar los datos, de tal modo que este objeto se encargará a su vez de informar a cada uno de los registradores del dato entregado por la RTU.

Su diseño está separado de la RTU para evitar la sobrecarga de registrar los datos dentro de la RTU, cuya única preocupación debería ser la obtención, decisión y distribución de los datos a las distintas clases.

El objeto registradores podría ser ampliado para enviar los datos en paralelo a cada una de las clases separando cada una de estas ejecuciones en hilos diferentes. No se descarta esta ampliación en la nueva RTU.

El objeto EVENTS Se creó para disparar eventos, es decir, a EVENTS le indicamos las condiciones y se le entregan los datos al igual que a los registradores. El objeto EVENTS está diferenciado de un registrador porque el camino de los datos es bidireccional, es decir, mientras que en un registrador los datos van siempre hacia el registrador, un objeto EVENTS puede devolver datos extras que provienen de toma de decisiones.

Un ejemplo de uso de EVENTS:

```
events=EVENTS(s)
events.addvariable("Uno", "1 TIME > 10s")
events.addvariable("Salida1", "BitA3 NOT")
events.addevent("Uno", controlador, "wr", (2, 5),
```

En este ejemplo se crea el objeto, se le añaden 2 variables y se le inserta un evento. Las variables añadidas son “Uno” y “Salida1”, que cambiarán su valor en función de las condiciones dadas:

- Uno - “BitA0 TIME >10s”: Será True cuando BitA0 esté a 1, más de 10s, será False en cualquier otro caso.

- Salida1 - "BITA3 NOT": Simplemente contendrá la negación de BITA3.

En el ejemplo anterior hemos ocultado un paso, que corresponde a la variable controlador. La variable controlador es el objeto de la RTU. Sí, es una realimentación como se puede ver en el siguiente ejemplo, ya completo:

```
events=EVENTS(s)
events.addvariable("Uno", "1 TIME > 10s")
events.addvariable("Salida1", "BitA3 NOT")
controlador=TELEMANDO("hola", registradores, events, s)
events.addevent("Uno", controlador, "wr", (2,5), "%Salida1")
```

donde "hola" es el nombre de la RTU, donde registradores es un objeto registradores, donde events es el objeto events ya declarado y donde "s" es el listado de señales obtenidas de senales.ods.

Efectivamente para declarar la RTU necesitamos el objeto EVENTS ya declarado, pero a su vez para insertar un evento necesitamos de la RTU. Esto es porque la RTU usará un objeto EVENTS para actuar, pero los eventos pueden reflejarse en otros registradores a su vez.

La línea del addevents se traduce en que si la señal "Uno" se pone a True, el evento se dispara y en el registrador se escribe en el relé (write relé = "wr") del slot 2 en la posición 5 (posicion = (2,5)) el valor contenido en la variable interna Salida1 ("%Salida1").

Un uso sencillo de la RTU El programa que haga uso de este sistema se encargará de hacer la ejecución en bucle y de definir la logística que deba tenerse en cuenta, además las RTUs basadas en Advantech soportan las funciones sencillas "PUT" y "GET" para facilitar el manejo de las señales.

Un ejemplo de ello podría ser el siguiente programa realizado para hacer pruebas sobre un Advantech ADAM 5000 TCP que tiene en cada slot:

- **Slot 1:** Entradas analógicas

- **Slot 2:** Salida Relés

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

# Importo librerías
from ADAM5000TCP import *
import time

# Creo una configuración por defecto
config=CONFIG()
config.port(502)
config.vpp(10)
config.dirlog(["src/tmp/log/dat/"])
config.cabecera_log("advantech")
#config.set_debug(True)

# Creo la RTU
```

```
a=TELEMANDO("hola",None,None,None)
a.config(config,"192.168.10.11")
#a.set_debug(True)
a.connect()

# Muestro todas las variables
print a.getall()

# Para siempre
while 0:

    if (a.get("ra", (1,2))%2):
        a.put("wr", (2,0),1)
    else:
        a.put("wr", (2,0),0)

    if (a.get("ra", (1,5))%2):
        a.put("wr", (2,1),1)
    else:
        a.put("wr", (2,1),0)

    if (a.get("ra", (1,6))%2):
        a.put("wr", (2,2),1)
    else:
        a.put("wr", (2,2),0)

    if (a.get("ra", (1,7))%2):
```

```
        a.put("wr", (2,3),1)
    else:
        a.put("wr", (2,3),0)

    if (a.get("rd", (2,4))):
        a.put("wr", (2,4),0)
    else:
        a.put("wr", (2,4),1)
```

```
# Desconecto
a.disconnect()
```

¿Y cómo hacer música con un Advantech ADAM 5000 TCP? Usando directamente la librería MODBUS_TCP y el siguiente código de ejemplo:

```
#!/usr/bin/env python
#-*- coding: iso8859-15 -*-

from lib.MATH_BIN import *
from lib.MODBUS_TCP import *
import sys
import time

# Musica
tempo=0.2
```

```
musica=[]
musica.append("00000000")
musica.append("10000000")
musica.append("01000000")
musica.append("00100000")
musica.append("00010000")
musica.append("00001000")
musica.append("00000100")
musica.append("00000010")
musica.append("00000001")
musica.append("00000010")
musica.append("00000100")
musica.append("00001000")
musica.append("00010000")
musica.append("00100000")
musica.append("01000000")
musica.append("10000000")
musica.append("10011001")
musica.append("10011001")
musica.append("01111110")
musica.append("01100110")
musica.append("10011001")
musica.append("10011001")
musica.append("01111110")
musica.append("01100110")
musica.append("10011001")
musica.append("10011001")
```

```
musica.append("01100110")
musica.append("10011001")
musica.append("01100110")
musica.append("00000000")
musica.append("11111111")
musica.append("00000000")
musica.append("11111111")
musica.append("00000000")
musica.append("10000000")
musica.append("01000000")
musica.append("00100000")
musica.append("00010000")
musica.append("00001000")
musica.append("00000100")
musica.append("00000010")
musica.append("00000001")
musica.append("00000010")
musica.append("00000100")
musica.append("00001000")
musica.append("00010000")
musica.append("00100000")
musica.append("01000000")
musica.append("10000000")
musica.append("10000000")
musica.append("01000000")
musica.append("01000000")
musica.append("00100000")
```

```
musica.append("00100000")
musica.append("00010000")
musica.append("00010000")
musica.append("00001000")
musica.append("00001000")
musica.append("00000100")
musica.append("00000100")
musica.append("00000010")
musica.append("00000010")
musica.append("00000001")
musica.append("00000001")
musica.append("10000001")
musica.append("01000010")
musica.append("00100100")
musica.append("00011000")
musica.append("00111100")
musica.append("01111110")
musica.append("11111111")
musica.append("11100111")
musica.append("11000011")
musica.append("10000001")
musica.append("11000011")
musica.append("11100111")
musica.append("11111111")
musica.append("11100111")
musica.append("11000011")
musica.append("10000001")
```

```
musica.append("00000000")

# Constantes
on=65280
off=0

# Connect
C=MODBUS_TCP("192.168.2.249")

# Send the messages and show answers
i=0
for frame in musica:
    try:
        print frame
        posicion=0
        for bit in frame:
            start=16+posicion
            if (bit=="1"):
                status=on
            else:
                status=off
        C.SEND(i,5,start,status)
        (id,function,data)=C.RECV()
        posicion+=1
    except Exception,error:
```

```
        print "ERROR: %s" % (error)
    time.sleep(tempo)
```

```
# Disconnect
C.DISCONNECT()
```

3.2.8. Un adelanto de la nueva RTU

La nueva versión de la RTU en la que estamos trabajando se ha desarrollado sobre una nueva filosofía de trabajo en la que tratamos de que el operador se centre más en los objetivos de la tarea y menos en cómo conseguir esos objetivos. La estructura interna de la RTU se ha diseñado para que equivalga a los PLCs más modernos sin perder las garantías de la propia filosofía de Likindoy: modularidad, sencillez y eficiencia.

También hemos incluido una extensa lista de mejoras:

- Nueva versión de los ficheros SIGNALS que permite realizar compresión binaria de los datos para reducir de forma drástica el espacio que ocupan en disco.
- Igualmente los ficheros SIGNALs v2.0 estarán ordenados y comprimidos internamente para acelerar la carga en el sistema principal.
- Dispone de variables compartidas de modo que sistemas externos pueden hablar en tiempo real con la

RTU para cambiar valores, leer valores que la RTU está usando tanto de entrada como de salida.

- Permite notificar en tiempo real de alertas del sistema por sms, email, jabber e incluso por los altavoces pronunciando el mensaje de alerta que indiqués.
- El sistema incorpora un autotest para asegurarse que no existen condiciones de posible bloqueo en el programa diseñado por el operador.
- Permite subir actualizaciones del programa en tiempo de ejecución y ponerlas en marcha de un modo sencillo, todo sin detener el funcionamiento de la RTU.
- El programa del operador puede controlar reinicios, apagados y actualizaciones.
- En la configuración de la RTU también se define el mapeado físico del sistema de tal modo que permita asegurarnos que las señales y sus posiciones son las que realmente hemos indicado en el programa y así evitar sorpresas.
- Los registradores seguirán funcionando del mismo modo que ya ocurre en la RTU actual.
- También estamos trabajando en implementar un sistema de Watchdog que asegure el control del sistema ante posibles problemas de estabilidad.

Un ejemplo de código para el operador sería:

```
# Procedimiento para tomar decisiones
def program(self):
    # Recupera variables
    exec(self._set_variables())

    # Si estamos comenzando
    if (self.starting()):
        # Declara las variables internas
        contador=1
        ADAM_BitB4=0
    else:
        # Incrementa
        contador+=1
        # Comprueba contador
        if (contador>=10):
            # Envía un email
            self.do("email", "juanmi@likindoy.com")
            # Indica que ha terminado pronto
            self.do("festival", "Programa RIFA que manda")
            # Termina y vuelve a la consola
            self.console()

    # Programa principal
    if (ADAM_BitB4):
        # Envía por UDP un dato
        self.do("udp", "ADAM_BitB4", 0)
```

```
# Cambia el valor a 0
ADAM_BitB4=0
else:
    # Envía por UDP un dato
    self.do("udp", "ADAM_BitB4", 0)
    # Envía por Jabber un mensaje
    self.do("jabber", "juanmi@likindoy.com")
    # Cambia el valor a 1
    ADAM_BitB4=1

# Memoriza las variables de la consola
return self._return_variables(log)
```

3.2.9. El servidor UDP

Durante nuestra explicación sobre la RTU, hemos hablado de que esta puede enviar paquetes UDP a un servidor externo. ¿Qué sentido tendría para nosotros disponer de una RIFA que manda paquetes si nadie los interpreta. Es para eso que se diseñó el servidor UDP que se encarga de cazar los paquetes y cargarlos en una base de datos local SQLite para que posteriormente sea procesada por algún programa e incluso nuestra versión genera un fichero PHP con el estado de las variables recibidas y que el mismo servidor mantiene actualizado en tiempo real, de este modo enlazar una página web a un servidor UDP se convierte en una tarea tan sencilla.

lla como insertar un "include" en el código PHP con el que estamos trabajando.

Un ejemplo de esto se puede ver en <http://tiempo.axaragua.com> donde un sistema HTML+AJAX pregunta a un sistema PHP sobre el estado de las estaciones meteorológicas y este gracias al "include" puede mostrar en cada consulta la información real.

Iniciar el servidor UDP es muy sencillo y pasa únicamente por ejecutar la función correspondiente:

```
launch(BUFSIZE, port, clave, outputfile, bd_nombre, knowledge, borrar_antiguos, acl, silent)
```

donde:

- **BUFSIZE:** tamaño del buffer.
- **port:** puerto en el que escuchar
- **clave:** clave de la conexión UDP
- **outputfile:** fichero en el que escribir los resultados
- **knowledge:** diccionario que contiene información útil para interpretar la información contenida en los paquetes
- **borrar_antiguos:** borra los registros antiguos cada n-ciclos. 60 ciclos equivalen a unos 5 minutos si cada ciclo ocupa 5 segundos.

- **acl:** es una lista de direcciones IPs de las que el servidor aceptará paquetes
- **silent:** indica si debe trabajar en silencio o no

Un ejemplo de la clave "knowledge" podría ser:

```
knowledge["plc1_sa_12"]="Nivel Oxigeno cuba A"
```

donde se indica que el paquete recibido como plc1_sa_12 (plc1, salida analógica, 12), se llama "Nivel Oxigeno Cuba A", que sus valores van de 0 a 10 y que es una señal analógica.

3.2.10. El sistema de estadísticas de red

Este sistema se diseñó para dar un uso menos industrial a Likindoy y aprovechar toda su potencia para hacer estudios del estado y evolución de las redes de datos de distintos servidores. Es un módulo que puede analizar tanto el ancho de banda de los diferentes interfaces como el retraso medio por envío de paquetes, en este último caso es necesario tener en funcionamiento un programa externo que se encarga de realizar las pruebas de retraso y exponer el resultado en un fichero al cual este módulo accede.

Al igual que el resto de módulos de Likindoy la declaración y uso es bastante sencilla, mostremos un ejemplo antes de explicar cada parámetro:

```
t=TELEMANDO("Trapiche", "data/red/netdata")
t.config(config)
t.addsignal("eth0",band_in)
t.addsignal("eth0",band_out)
t.addsignal("192.168.1.1",ping_digital)
t.addsignal("192.168.1.1",ping_digital)
t.addsignal("192.168.1.1",ping_maximo)
t.addsignal("192.168.1.1",ping_minimo)
t.addsignal("192.168.1.1",ping_medio)
```

donde las variables predefinidas son:

```
band_in="bi"           # Bandwidth input (ancho de banda de entrada)
band_out="bo"         # Bandwidth output (ancho de banda de salida)
packet_in="pi"        # Packages input (paquetes de entrada)
packet_out="po"       # Packages output (paquetes de salida)
ping_digital="hp"     # Hay ping?
ping_enviados="pe"   # Pings enviados
ping_recibidos="pr"  # Pings recibidos
ping_perdidos="pp"   # Ping perdidos (%)
ping_minimo="pm"     # Ping minimo
ping_medio="pa"      # Ping average (ping medio)
ping_maximo="px"     # Ping máximo
ping_dispositivo="pv" # Ping dispositivo (mdev)
```

Viendo el ejemplo es sencillo entender que primero se crea el objeto (dando el nombre interno y el fichero de salida),

después se le pasa la configuración por defecto y finalmente se añaden las señales que se vayan a usar dando como primer parámetro la señal afectada y como segundo grupo el "conector" de esa señal (el dato de esa señal que deseamos trabajar).

Para llevar el control de pings deberemos tener lanzado el comando "pinger.py" que se encargará de preparar las estadísticas de pings según su configuración en la que simplemente especificaremos un listado de direcciones IPs o hosts a las que se les realizarán los tests necesarios.

3.3. Manual de administrador

La documentación para administradores es muy escueta ya que la gran mayoría de ella corresponde a la instalación ya explicada anteriormente. Los administradores y desarrolladores podrán encontrar los aspectos más técnicos de la documentación en el directorio doc/html/index.html del programa.

3.3.1. Debuging del programa

Inicialmente todos los comandos al lanzarlos activan el debuging del programa, de tal modo que aparecerá información útil en pantalla acerca del proceso en ejecución. Para ocultar el debuging simplemente hay que añadir "--silent" al comando en cuestión.

3.3.2. Usando el cron

En nuestros sistemas las tablas de CRON han quedado así:

```
# Generación de gráficas
0 6 * * * /home/user/likindoy/cron.sh --start --silent
0 7,8,9,10,11,12,13,14,15,16,17 * * * /home/user/likindoy/cron.sh --continue --silent
0 18 * * * /home/user/likindoy/cron.sh --finish --silent
# Envío de emails
45 13,14 * * * /home/user/likindoy/cron_emails.sh --silent
15 7,8,9,11 * * * /home/user/likindoy/cron_emails.sh --silent
```

El programa se inicia con "--start" a las 6:00am lo que inicia el proceso de descarga, carga a la base de datos y generación de gráficas. Desde las 7:00am y hasta las 17:00, cada hora se le recuerda al sistema que siga trabajando ("--continue"), con ello conseguimos que si a cierta hora no se pudieron bajar los datos de un Telemando concreto, el sistema vuelva a reintentarlo más tarde y genere sus graficas también. Ojo sólo descarga, carga y generación, pero no envío de gráficas. A las 18:00 le indicamos al sistema con "--finish" que debe terminar su ejecución y para ello debe generar y enviar todas las gráficas tal cual estén.

Independientemente durante todo el día, de las gráficas generadas las enviará por email a las horas indicadas: 7:15am, 8:15am, 9:15am, 11:15am, 13:45pm y 14:45pm.

El sistema produce salidas de error que debes controlar en "cron.sh" y "cron_emails.sh" o bien directamente en "crontab" indicando el email por defecto al que enviar las salidas del cron.

3.3.3. Conocer el estado actual

Para conocer el estado actual puedes consultar con el comando: **watch -n 0 "ps axf | grep py | grep -v grep"**, el estado del sistema para ver que módulos se están ejecutando. Independientemente la nueva versión de Likindoy (aún no publicada) contiene un sistema de logs que irá informando directamente sobre ellos del tiempo que ha tardado cada cosa y la hora en que se realizó la acción.

Si ves que Likindoy no está funcionando, lánzalo a mano para conseguir así el candado de bloqueo del sistema y ver que ocurre durante la ejecución. Por ejemplo, si lanzas el "adquirir_signals.py" entonces el script principal ya no podrá hacerlo y serás tú el que verá en pantalla (gracias al debugging) lo que el sistema está haciendo.

3.3.4. El funcionamiento

Likindoy tiene 4 etapas básicas de funcionamiento, además de las ampliadas por el servidor UDP y otros:

1. **Adquisición de datos:** el sistema se conecta a todas las RTU (unidades remotas) para descargarse los datos

de cada una.

2. **Carga de datos:** procede a cargar toda la información obtenida en las bases de datos del programa.
3. **Generación de gráficas:** usa la información disponible en la base de datos para generar las gráficas que le sean posibles en función del test de validación asignado a cada gráfica.
4. **Envío por email:** usando todas las gráficas generadas se envía un email a cada usuario de la lista si sus gráficas estaban disponible (debemos tener en cuenta que cada usuario puede recibir un grupo distinto de gráficas).

3.3.5. Desarrolladores de Likindoy

Para ampliar Likindoy deberás acudir a la documentación de Doxygen, mira el apartado Doxygen de este mismo document.

Independientemente el proceso que recomendamos para generar un nuevo módulo es copiar un módulo ya existente con un nuevo nombre y después eliminar los métodos que no se vayan a usar y dejar el formato de la clase NOMBRE_MODULO_CONFIG y de la clase NOMBRE_MODULO, dejar toda la parte del constructor y carga de la configuración por defecto, así como todos los métodos que deseemos tener por compatibilidad. No olvides duplicar

los ficheros de configuración y tampoco los de inicio del programa que puedes encontrar en carpeta raíz de Likindoy.

En general aprende como generar un nuevo módulo mirando módulos ya existentes y no olvides visitar los foros de Likindoy para solucionar tus dudas.

4. Detalles del sistema

4.1. Análisis funcional

Desde el punto de vista funcional, Likindoy supone un gran avance en el área de informática industrial, debido principalmente a dos factores muy importantes: **acerca los sistemas industriales a los Ingenieros Informáticos y los costes de implantación son muy reducidos.**

4.1.1. Alcance funcional

Likindoy puede trabajar en un entorno mixto donde se atiende a la vez a PLCs, RTUs de Likindoy, PCs conectados a la red, clientes UDP y unidades remotas, además la misma máquina puede hacer la generación de datos y publicación de los mismos (por web, por email, por UDP u otros protocolos).

Likindoy puede cubrir con éxito el seguimiento de históricos, esto significa que puede generar gráficas o compartir

datos en tiempo real puesto que pasado 1 segundo del momento actual ya se considera un histórico para Likindoy y esto se puede extender en el tiempo hasta centenares de años si es necesario. Obviamente la obtención, carga de datos y generación en la base de datos debería ser más rápida de dicho segundo, lo cual no es viable en la generación de gráficas aunque sí lo es en la compartición de datos por UDP, WEB u otros. Además el sistema de RTU permite tomar decisiones con tiempos de retraso no superiores a 1 segundo (dependiendo del retraso de la red y del número de señales procesadas).

Likindoy puede ser usado en cualquier entorno industrial que requiera un seguimiento del estado de los sistemas de un modo parcial, es decir, qué ocurrió en las últimas 4 horas, que ocurrió ayer, etc...y usar esta información para realizar previsiones de mantenimiento y ordenar el trabajo de los operarios en consecuencia con la información obtenida. Igualmente permite conocer qué parámetros ajustar para optimizar los medios de producción y además ayuda a conocer de un vistazo los efectos laterales de ciertas acciones que no veríamos en el proceso en sí, es decir, si llueve el humo producido por la planta se condensa antes en la salida produciendo una lluvia interna que provoca un aumento general de la presión en la etapa de salida.

Igualmente Likindoy puede ser usado en sistemas de servidores para controlar información de la red de tal modo que sea posible conocer cuando un sistema está consumien-

do más Internet de lo que debería y es posible atender a un seguimiento para encontrar las causas, situación imposible de descubrir si el evento ocurre ocasionalmente y que quedaría patente en una gráfica.

Incluso puede ser ampliado, cualquier programador con unos conocimientos básicos de Python puede ampliar Likindoy con nuevos módulos para atender a otros sistemas usando otros protocolos.

Además Likindo-RTU permite un control en tiempo real para acciones y reacciones que requieren tomar decisiones en el instante y de un modo automático (sin la decisión de un operador).

4.1.2. Casos de éxito

Gracias a Axaragua, Likindoy se ha ido completando con un amplio abanico de casos de éxito que confirman la funcionalidad y estabilidad del sistema.

- **Gráficas diarias:** todos los días desde comienzos del año 2006 estamos recibiendo gráficas diaras de los sistemas de Axaragua y nos ha ayudado a prevenir situaciones de riesgo, roturas de motores y plantear los mantenimientos diarios. Tanto gerente, como encargados de planta, operarios e incluso clientes reciben las gráficas de Likindoy, de tal modo que cada cual actúa en consecuencia con sus responsabilidades.

- **Likinweather:** nos permite mostrar información del estado actual de la meteorología en tiempo real, lo mejor para explicarlo es verlo: <http://tiempo.axaragua.com>.
- **Análisis de red:** igualmente tanto de Axaragua como de Centrológic recibimos gráficas del flujo de la red en ellas podemos ver si una conexión está saturada por exceso de uso, de donde viene la carga que se produce, cómo se comporta durante la noche, etc... Una información muy útil para administradores de grandes redes.
- **Detección de eventos esporádicos:** en una ocasión Likindoy nos permitió detectar y conocer más de cerca las razones por las que unas válvulas se abrían y cerraban esporádicamente con una duración aproximada de un segundo. El evento ocurría unas 3 veces cada 10 minutos por lo que verlo haciendo un seguimiento en tiempo real sería impracticable. En nuestro y gracias a Likindoy pudimos observar el evento directamente y compararlo con otras señales para detectar la causa de este curioso efecto.
- **Gráficas en la web:** además Likindoy nos permite mostrar gráficas en la web para que los visitantes puedan conocer como varió el tiempo ayer o consultar todo el histórico de los pantanos de agua hasta la fecha de

hoy.

4.1.3. Advertencia

Likindoy-RTU se ha diseñado para ser muy estable, pero no deja de ser un software bastante reciente por lo que requiere un periodo más amplio de pruebas y un abanico más amplio de entornos de producción.

Es por ello que te invitamos a que incluyas Likindoy como un sistema añadido de apoyo en tu entorno de producción que no interfiera en la producción pero sí genere un valor añadido a los datos recogidos.

En nuestro caso el sistema funciona perfectamente en los entornos de producción en los que ha sido probado y los resultados que produce son de una calidad muy alta, es por ello que hemos decidido compartir el proyecto.

Este es código e información pre-alpha. Usted asume toda la responsabilidad por su uso.

Aunque Likindoy en la actualidad toma datos y no da órdenes, con un poco de experiencia se puede conseguir que de órdenes por ello hacemos la siguiente advertencia:

PELIGRO: ¡¡¡NO conecte a un PLC a menos que es hace!!!. Asumimos que usted tiene experiencia y resolución de problemas en los PLC, y que usted sabe que esta haciendo. Los PLCs se usan para controlar industriales, motores, válvulas de vapor, presiones. Usted es TOTALMENTE RESPONSABLE de asegurarse

peligro de accidentarse o morir porque usted altere los **datos del** **adquisitor web** que ejecuta un PLC.

4.2. Diseño técnico

4.2.1. Diagrama principal del sistema

Adquisición de datos - Cargador - Generador de gráficas
- Envío por emails

4.2.2. Detalles del árbol

Likindoy se organiza como se indica a continuación

- data: datos del sistema.
 - **dib**: imágenes generadas por el generador de gráficas.
 - **dib.bak**: copia que no se borra de las imágenes generadas por el generador de gráficas.
 - **log**: el sistema logs guardará aquí los detalles de lo que vaya ocurriendo.
 - **red**: datos del adquisitor de red.
 - **reg**: ficheros de registros obtenidos durante la adquisición.
 - **reg.bak**: copia que no se borra de los ficheros de registros obtenidos durante la adquisición.

- doc: documentación.

- **changelog.gz**: fichero que informa de los cambios producidos en Likindoy.
- **configuraciones**: documentación extra acerca de los ficheros de configuración (merece la pena leerla).
- **faq**: preguntas y respuestas a problemas poco comunes pero importantes.
- **html**: aquí reside la documentación generada por Doxygen, comienza en index.html.
- **install.py**: script de la instalación de Likindoy, comprueba que el sistema dispone de todas las librerías necesarias y además permite configurar la base de datos y crear las estructuras necesarias para unirse con Likindoy.

- etc: contiene todas las configuraciones que el operador de Likindoy debe manejar.

- src: código fuente

- **actuadores**: contiene lo que nosotros denominamos “actuadores”, son partes del sistema de Likindoy que se encargan de “actuar” o realizar acciones concretas que son finales y definitivas de

cara al sistema. Aquí podrás encontrar los sistemas para: enviar las gráficas por email, informar mediante Jabber de ciertos eventos o incluso pronunciar mediante un sintetizador de voz por los altavoces (gracias a “Festival”) de lo que está ocurriendo.

- **cargadores:** en la actualidad existen un único cargador dentro de este directorio y es el que se encarga de transferir los ficheros de `data/reg/` a la base de datos MySQL que se le indique. Su nombre “`OUT_DB_MYSQL`” indica que el sistema recogerá en la entrada un fichero de datos y en su salida pasará este a una base de datos MySQL. En este directorio se deben encontrar únicamente aquellos sistemas que se encarguen de recoger ficheros adquiridos por Likindoy y pasarlos en su salida a otro sistema diferente (ficheros, servidores, bases de datos, ...). Son meros transformadores de formato.
- **etcbin:** contiene toda la configuración de Likindoy tanto los ficheros de configuraciones por defecto y las configuraciones ocultas del operador, como las configuraciones visibles del operador. Todo está en este directorio. A modo informativo se indica que las configuraciones del operador son enlaces al directorio `etc/` de Likindoy.
- **graficas:** contiene módulos de entradas y mó-

dulos de salida, estos módulos usan la clase `lib/GRAFICAS.py` para convertirse en generadores de gráficas mediante herencia de esta. Los módulos de entrada sirven de apoyo como interfaz de entrada de datos y los módulos de salida son los que producen las gráficas en los formatos que se estimen oportunos. En este directorio encontramos 3 módulos:

- IN_DB_MYSQL: usa una base de datos MySQL como interfaz de entrada de datos.
- OUT_DB_MYSQL: usa una base de datos MySQL para alimentar esta con los datos recogidos. El sentido de este módulo es permitir la realimentación de los datos de nuevo al sistema. Con este módulo podemos generar nuevas señales en la base de datos a partir de señales ya existentes, al usar la potencia del módulo heredado `GRAFICAS` podemos usar la calculadora completa de Likindoy para computar unas señales con otras y cargarlas en la base de datos. Es muy útil para realizar estadísticas de los datos almacenados en la base de datos.
- OUT_R: usa el sistema R para generar gráficas mediante este lenguaje. El resultado es expresado en PDF, JPG o PNG. PNG es la configuración que nosotros usamos por de-

fecto, por ser de mayor calidad que el JPG y más útil que el PDF.

- **lib:** es el directorio donde podemos encontrar todas las librerías que usa Likindoy, está bastante completo y si desea conocer más acerca de cada librería simplemente acuda a la documentación avanzada (Doxygen) en el directorio `doc/html/index.html`
- **telemandos:** contiene todos los módulos con los que Likindoy puede obtener datos que después procesará, aquí están mezclados todos los “adquisidores de datos”, es decir, cualquier sistema que pueda obtener datos de un sistema externo. Siguiendo esta filosofía están los módulos:
 - ADAM5000TCP: permite obtener datos de sistemas de adquisición de datos Advantech ADAM 5000 TCP y almacenarlos en un fichero de datos, facilita en gran medida las conversaciones con estos sistemas. Se usa para adquisición en tiempo real, permite la toma de decisiones y envío de alertas.
 - FILE_FTP: permiten obtener ficheros de datos de servidores FTP. El fichero descargado ya es un fichero datos utilizable por Likindoy.
 - FILE_SFTP: permiten obtener ficheros de

datos de servidores SFTP. El fichero descargado ya es un fichero datos utilizable por Likindoy.

- MOMENTUM: permite obtener datos de sistemas Schneider Momentum y almacenarlos en un fichero de datos. Se usa para adquisición en tiempo real.
- NETSTATISTICS: permite obtener datos de los sistemas de red del equipo en cuestión y almacenarlos en un fichero de datos. Se usa para adquisición en tiempo real.
- PESYR_FTP: permite obtener datos de sistemas PESYR mediante FTP, usando el protocolo de pregunta/respuesta de PESYR mediante FTP. Es un proceso lento y no informado por parte del servidor de PESYR. El fichero descargado ya es un fichero datos utilizable por Likindoy.
- SIGNALS_FTP: permite obtener datos de RTUs de Likindoy mediante extracción de los datos por FTP y usando el protocolo de pregunta/respuesta definido por Likindoy-RTU. El fichero descargado ya es un fichero datos utilizable por Likindoy.
- WEB: permite obtener datos directamente de una web y almacenarlos en un fichero de datos. Se usa para adquisición en tiempo real.

- **tmp:** directorio que almacena la información temporal de Likindoy mientras este está trabajando.

4.2.3. Lenguajes de programación usados

Likindoy está realizado íntegramente con tecnologías Open Source:

- El sistema operativo es **Linux**
- El lenguaje de programación es **Python**
- El gestor de bases de datos es **MySQL**
- Para la generación de gráficas se usa el lenguaje de programación **R**, sucesor del lenguaje **S** e ideado esencialmente para uso estadístico y generación de gráficas.

¿Por qué usar Linux? Las razones para usar Linux son demasiadas para exponerlas todas, por lo que nos centraremos en las principales cualidades que nos hicieron optar por este sistema operativo:

1. **Estabilidad:** porque el sistema no se desconfigura sólo, no sufre de bloqueos generales, si un proceso se bloquea lo terminas, revisas por qué pasó y lo inicias de nuevo sin afectar a la estabilidad del sistema y sin dejar de trabajar.

2. **Seguridad:** principalmente Linux “no soporta” virus y aunque existiesen no afectarían al sistema por el modo en que éste ha sido diseñado.
3. **Gratuito:** el sistema operativo es fácil de encontrar en cualquier lugar, no es ilegal descargarlo, instalarlo o usarlo.
4. **Libre:** puedes encontrar el código fuente de Linux disponible en Internet por lo que podrás reparar algo si tienes el conocimiento para hacerlo.
5. **Vivo:** es un sistema operativo en continuo crecimiento, puedes descargarte actualizaciones y mejoras casi diariamente si lo deseas.

¿Por qué usar Python? Muchos desarrolladores indicarán que un programa de estas características debería estar en un lenguaje compilado y no en un lenguaje interpretado como Python. La razón por la que nos decantamos por Python fue esencialmente porque mantener el código es mucho más sencillo que en un lenguaje compilado. Si el programa falla en el lado del servidor siempre puedes editar el código fuente y reparar, ya que el código fuente está donde está el programa. Otra razón muy importante es la inmensa cantidad de librerías disponibles para Python así como la documentación para cada una de ellas.

¿Por qué usar MySQL? La decisión fue bastante sencilla, en Linux existen dos gestores importantes de Bases de Datos SQL, esto son: MySQL y PostgreSQL. PostgreSQL es más difícil de iniciar y no dispone de tanta herramientas como MySQL, en añadido MySQL es más rápido en los procesos de lectura que PostgreSQL. Igualmente y aunque es bien conocido que PostgreSQL es el más estable de los dos, MySQL ha sufrido importantes avances en los últimos años que lo han convertido en el servidor de Bases de Datos favorito por la mayoría de los servidores de Internet.

¿Por qué usar R? R es empleado ampliamente por la comunidad científica para generar gráficos, es configurable, libre y potente.

4.3. Modelos de datos

Los datos en Likindoy se almacenan de un modo mixto dependiendo del área de trabajo existe un modelo de datos para así optimizar su funcionamiento y facilitar su so. De este modo los diferentes modelos de datos vienen explicados a continuación:

4.3.1. El fichero de señales

El fichero de señales usa el formato OpenDocument Spreadsheet (Hoja de cálculos de OpenDocument), que se

explicó anteriormente en el punto 3.2.4. Es un fichero XML que contiene la información necesaria con el listado de señales en filas y las opciones de cada señal en columnas.

4.3.2. Configuraciones

Las configuraciones son un formato reducido de las declaraciones de objetos y el uso de métodos con Python. El objetivo es que Likindoy provea un objeto final sencillo de usar y que contiene los métodos que el usuario necesita, de este modo una gráfica es un objeto que permite definir aspectos como los ejes o las líneas que va a incluir, y un telemando se basa en su forma de acceso y el formato de los datos.

De este modo, generalmente los objetos son creados por el operador en el mismo fichero de configuración y enlazados a una lista, en el caso de las graficas el operador generará un objeto gráfica, lo configurará mediante los métodos y finalmente incluirá este objeto en el listado de gráficas que posteriormente será procesado por el scrip generador de gráficas y del que el operador no debe tener conocimiento de cuando se ejecuta y cómo se va a ejecutar.

El disponer de los ficheros de configuración en Python nos permite además realizar una selección de configuraciones en función del modo de funcionamiento. Ejemplo: si existe el fichero xyz.dat en el disco duro entonces incluye este grupo de gráficas en el generados, en caso contrario in-

cluye este otro grupo, el permitir configuraciones bajo Python ofrece una flexibilidad muy alta a la hora de escribirlas y trabajar con ellas.

4.3.3. Ficheros PESYR-INFOCEL

Los ficheros de INFOCEL siguen un formato actualmente abierto pero que es propietario de la empresa PESYR. El formato de fichero cumple que en cada línea se muestra una captura completa del sistema y su situación actual de tal modo que ahí queda reflejada toda la información en ese instante. Es un fichero tipo texto legible por cualquier editor que separa cada registro por filas y cada campo de registro por columnas.

El formato concreto de este tipo de ficheros puede ser consultado directamente en el código fuente de Likindoy o bien solicitando información ampliada a PESYR.

4.3.4. Ficheros SIGNALS

Los ficheros de tipo SIGNALS son un formato propio de Likindoy que cumple las características de ser abierto y libre. Este formato de ficheros registra en cada fila una señal y su estado correspondiente, de tal modo que en cada fila encontraremos una señal u otra y la captura instantánea de la misma. Existen dos tipos de filas “digitales” y “analógicas”:

- **Digitales:** contienen un total de 5 campos separados como se indica a continuación:
 1. Fecha en formato texto
 2. Fecha en formato epoch
 3. Función
 4. Posición
 5. Valor actual

- **Analógicas:** contienen un total de 9 campos separados como se indica a continuación:
 1. Fecha en formato texto
 2. Fecha en formato epoch
 3. Función
 4. Posición
 5. Valor actual
 6. Valor medio
 7. Valor máximo
 8. Valor mínimo
 9. Varianza

Como se indicó anteriormente los ficheros SIGNALS van a sufrir algunas mejoras en su nueva versión, de tal modo que sea posible comprimir la información contenidos en

ellos a un formato binario que ocupe menor espacio en disco.

4.3.5. Bases de datos SQL

En las bases de datos disponemos de 2 tablas que almacenan la información necesaria para que el generador de gráficas funcione adecuadamente.

■ **Tabla ids:**

- id: identificador de la señal cuyos datos podemos encontrar en la tabla de datos.
- md5: texto md5 resultado de hacer el md5 del nombre de la señal en texto.
- updated: indica la fecha epoch de la última actualización de dicho campo.

■ **Tabla datos:**

- id: identificador de la señal que podemos encontrar relacionado en la tabla ids.
- server: fecha del servidor en el momento de la inserción.
- fecha: fecha del programa en el momento de la inserción.
- hora: hora del programa en el momento de la inserción.

- valord: valor digital si la señal es digital.
- valora: valor analógico si la señal es analógica.

5. Doxygen

Likindoy ha sido programado usando la tecnología Doxygen para documentar el código fuente del programa. Para leer la documentación actualizada sólo tiene que acudir al directorio `doc/html/index.html` y en ella podrá encontrar cada una de las funciones del programa. La documentación que más útil le resultará la puede encontrar en "Módulos/MATH_GRA" para información referente a la calculadora de señales (resolución de ecuaciones), "FILTER" para conocer las traducciones de texto en los nombres de ficheros y nombres de las gráficas y para el uso de "addaxis" y "addline" podrá encontrar documentación en "Lista de clases/likindoy.src.lib.GRAFICAS.GRAFICADOR".

El código fuente de Likindoy se inició en español y se ha continuado en inglés para hacerlo más universal, no obstante aunque algunas clases se han traducido, otras aún continúan en español. Poco a poco iremos traduciendo el código a inglés en función de las necesidades de los usuarios de Likindoy.

Igualmente este documento está en español, pero en breve realizaremos una versión en inglés igualmente.

6. Licencia

Likidoy ha sido creado y es mantenido por, Juan Miguel Taboada Godoy y Juan José Denís Corrales. Protegido bajo la licencia GNU/GPL versión 3 que puede consultar [en su versión original](#) o en [en su versión no oficial traducida al castellano](#).

La versión actualizada de la librería MODBUS_TCP ha sido creada por Michael Griffin.

7. Conclusiones

Likidoy supone un gran avance en el entorno industrial porque ofrece un sistema abierto, modular, genérico, intuitivo, escalable y gratuito pensado para procesos industriales y cualquier tipo de proceso que se asemeje a estos. Está pensado para que los informáticos puedan integrarse en él con bastante facilidad y así acercarlos rápidamente a los procesos industriales permitiéndoles sacarles todo el partido desde el punto de vista de la programación y las comunicaciones.