ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

Elaboración de una interfaz de comunicación para módulos domóticos

Asier Ortés Alfonso

Tutor: Ignacio Raúl Matías Maestro

Pamplona, Septiembre de 2010

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

Elaboración de una interfaz de comunicación para módulos domóticos

ÍNDICE GENERAL

Asier Ortés Alfonso

Tutor: Ignacio Raúl Matías Maestro

Pamplona, Septiembre de 2010

Índice

1. Memoria
2. Diagramas
3. Pliego de condiciones
4. Presupuesto
5. Bibliografía
6. Anexos

Pamplona, Septiembre 2010

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

Elaboración de una interfaz de comunicación para módulos domóticos

MEMORIA

Asier Ortés Alfonso

Tutor: Ignacio Raúl Matías Maestro

Pamplona, Septiembre de 2010

Índice

1 Introducción 9

1.1 Objetivos y motivación 9

1.2 Módulos domóticos 11

1.3 E.commander 13

1.3.1 Introducción 13
1.3.2 Configuración 14
1.3.3 Captura de los valores 15

1.4 La aplicación 17

2 Primeros Pasos 19

2.1 Estudio de la comunicación 19

2.1.1 Introducción 19
2.1.2 Wireshark 19
2.1.3 Primeros paquetes de información 20

2.2 Transmisión FTP 21

2.2.1 Conexión 21
2.2.2 Transmisión de información sobre el sistema 22
2.2.2.1 El archivo #actual.sta 22
2.2.2.2 El archivo #summary.sta 23
2.2.3 Desconexión FTP 26

2.3 Transmisión de los datos TCP 28

2.3.1 Introducción a TCP 28
2.3.2 El segmento TCP: La Cabecera 29
2.3.3 El segmento TCP: Datos 31
2.3.3.1 Cabecera UDBF 31
2.3.3.2 Caracteres de separación UDBF 35
2.3.3.3 Datos UDBF 35
2.3.3.3.1 Interpretación de datos IEEE 754 35
2.3.3.4 Checksum UDBF 36
2.3.4 Transmisión de datos real 37

2.3.4.1 Análisis: Primer segmento 37
2.3.4.2 Análisis: El segmento de datos 38
2.3.4.2.1 Cabecera TCP 38
2.3.4.2.2 Comparación con estándar UDBF 39
2.3.4.2.3 Conversión real de valores 40
2.3.4.2.4 Otra información 40
2.3.4.3 Análisis: Segmentos siguientes 42
2.3.4.4 Desconexión TCP 44

3 Introducción a C# 46

3.1 Conceptos básicos 46

3.2 Ejemplo 1: Hola Mundo 47

3.3 Ejemplo 2: Llamando a métodos 49

3.4 Ejemplo 3: Estructuras condicionales, bucles y Arrays de datos 51

4 Solución adoptada 54

4.1 Bienvenida 54

4.1.1 Descripción 54
4.1.2 Propiedades 54
4.1.3 Métodos 54
4.1.4 Aspecto final 55

4.2 Menú Principal 56

4.2.1 Descripción 56
4.2.2 Propiedades 56
4.2.3 Métodos 56
4.2.4 Aspecto final 57

4.3 Configuración 58

4.3.1 Descripción 58
4.3.2 Diseño 58
4.3.2.1 Controles y cuadros de texto 58
4.3.2.2 Botones 62
4.3.2.2.1 Menú 62
4.3.2.2.2 Ayuda 62
4.3.2.2.3 Descargar 63
4.3.2.2.4 Actualizar 64

4.3.2.3 Carga de imágenes y aspecto final 69
4.4 Captura 70

4.4.1 Descripción 70
4.4.2 Diseño 70
4.4.2.1 Conexión y descarga 72
4.4.2.2 Procesado de los valores 73
4.4.2.3 Representación 75
4.4.3 Aspecto final 78

4.5 Gráficos 79

4.5.1 Descripción 79
4.5.2 Diseño 79
4.5.2.1 Formulario Windows 79
4.5.2.1.1 Botón Volver a Menú 79
4.5.2.1.2 Botón Ayuda 80
4.5.2.1.3 Botón Crear Gráfica 80
4.5.2.1.4 Selección de fechas 81
4.5.2.1.5 Máximo y mínimo 82
4.5.2.1.6 Llamada a la representación 83
4.5.2.2 El control para representación gráfica ZedGraph 83
4.5.3 Aspecto final 89

4.6 Ayuda 90

4.6.1 Descripción 90
4.6.2 Aspecto final 90

4.7 Vídeo 91

4.7.1 Descripción 91
4.7.2 Diseño 91
4.7.3 Aspecto final 92

4.8 Program 93

4.8.1 Descripción 93
4.8.2 Diseño 93
4.8.2.1 La clase FTPFactory 93
4.8.2.2 Método Login 94
4.8.2.3 Método getFileList 95
4.8.2.4 Método Download 95
4.8.2.5 Método Upload 96
4.8.2.6 Método Close 97

5 Conclusiones 98

1 Introducción

1.1 Objetivo y motivación

El objetivo con el que nació este proyecto fue desarrollar una interfaz de comunicación para módulos domóticos del modelo E.Reader del fabricante Gantner Instruments. El cometido de esta interfaz sería el de interpretar los datos ofrecidos por el módulo y entregarlos a una interfaz gráfica para su representación. Una vez cumplido dicho objetivo se añadió una segunda tarea, la de diseñar e implementar la interfaz gráfica. Estas dos interfaces unidas crean una aplicación que debe ser capaz de trasladar datos desde el módulo y presentarlos de tal manera que un usuario pueda interpretarlos, así como enviar información especificada por el usuario al E.Reader, es decir la comunicación debe ser bidireccional. Otro de los objetivos del proyecto es comprobar si es posible adaptar estos módulos para su manejo mediante esta interfaz de control integral, y en caso de no ser posible, discernir cuales son las limitaciones de este sistema. A la hora de crear la interfaz de usuario, se valorará el hecho de que esta sea intuitiva y sencilla, pero que a su vez permita realizar todas las funciones necesarias.

Para llevar a cabo el proyecto se dispone de un módulo E.Reader, con los accesorios necesarios para su correcto funcionamiento y manejo, como una fuente de alimentación y un armario con raíles DIN para montaje. También se dispone de un módulo adicional de entradas y salidas del mismo fabricante modelo E.Bloxx, que consta de 8 Entradas/Salidas configurables. Se cuenta asimismo con varios sensores para medición de parámetros como la temperatura ambiente o la presión atmosférica. El E.Reader tiene como complemento un software, llamado E.Commander que realiza las funciones que se pretenden replicar en este proyecto y por lo tanto, es necesario estudiar su funcionamiento dentro de lo posible para basarse en él y poder mejorarlo.

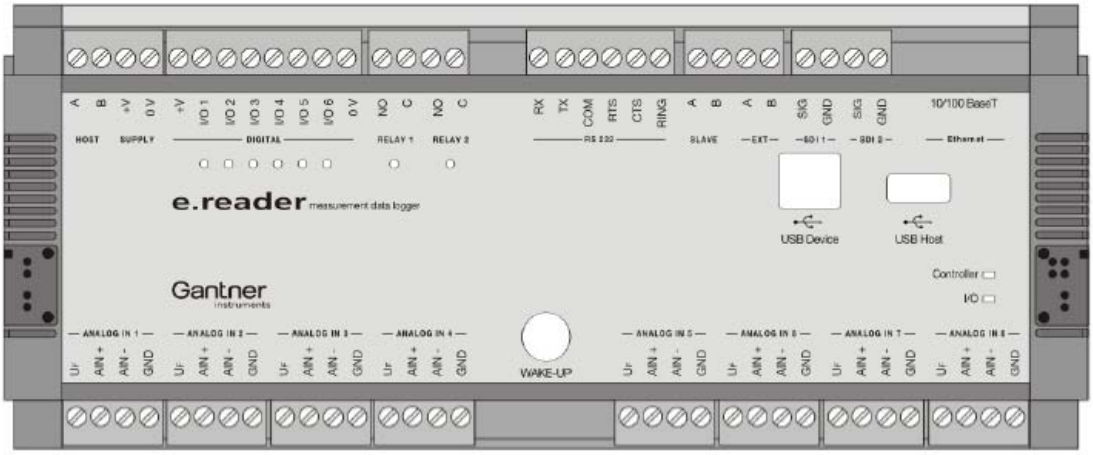
Además de estas necesidades de material, también es necesario tener conocimientos de algún lenguaje de programación de alto nivel, ya que son los más adecuados para la creación de interfaces de usuario. También es importante conocer los principales protocolos de comunicación y su funcionamiento. Dado que, en este caso, no se tienen conocimientos previos de lenguajes de alto nivel ni de creación de interfaces, habrán de adquirirse mediante libros y tutoriales Web.

En principio no es necesario que la interfaz resultante de este proyecto esté enfocada a ninguna aplicación concreta, por lo que podrá utilizarse en variedad de ellas, desde control de cultivos en explotaciones agrícolas a monitorización de parámetros médicos en hospitales. Para dotar de un aspecto visual más atractivo a dicha interfaz y dado que los módulos E.Reader están diseñados haciendo énfasis en aplicaciones industriales o en condiciones duras, se ha elegido como tema una explotación agraria, sin embargo las diferencias entre una aplicación y otra serán estéticas, por lo que la adaptación solo dependerá de los sensores que se conecten al módulo.

Esta memoria está dividida en cinco partes. Una primera de antecedentes e introducción, donde se sentarán las bases sobre las que comenzar el proyecto. La segunda corresponde con la fase inicial del proyecto, en la que se estudiará como realizar las conexiones y los resultados de las mismas. En la tercera parte se detallará el proceso de aprendizaje del lenguaje de programación con el que se implementará la interfaz, así como el impacto sobre el diseño de la misma. En la cuarta parte se detallará la solución adoptada. Por último, la quinta parte mostrará las conclusiones del proyecto.

1.2 Módulos domóticos

El módulo domótico para el que se creará la interfaz es el E.Reader de Gantner Instruments. El E.Reader es un módulo de adquisición de datos, habitualmente utilizado en entornos industriales, especialmente diseñado para la medida de señales eléctricas y datos térmicos o mecánicos en entornos experimentales.



Este módulo consta de 8 entradas analógicas a las que se puede aplicar algunos procesos como comparación y funciones matemáticas, así como almacenar los datos que entregan. La precisión de las entradas analógicas es de entre 0,01% y 0,5% dependiendo del rango de medida. Este dato es importante, pero para calcular el error máximo final que obtendremos con respecto al valor real, hay que tener en cuenta también la precisión propia del sensor que realiza la medida. Esta misma unidad, incorpora a su vez 6 entradas/salidas digitales para activación de alarmas y umbrales, además de 2 salidas de tipo relé. Los datos que se conviertan de analógicos a digitales utilizando el conversor interno del módulo tendrán una resolución de 16 bits, con una frecuencia de muestreo máxima de 1Hz y utilizando el tipo de conversión Sigma Delta. Para almacenar datos, el equipo consta de una memoria tipo flash interna de 128 MBytes y memoria RAM de 8 MBytes.

En cuanto a las opciones de comunicación, el E.Reader cuenta con varias interfaces, entre las que se encuentran un puerto RS 485, para conexión con otros módulos de la marca, como podría ser otro E.Reader o módulos de expansión tipo E.Bloxx. Para conexión externa existen dos posibilidades, o bien mediante un puerto serie RS 232, en cuyo caso se utilizará el protocolo SDI-12, que está diseñado específicamente con módulos como el E.Reader en mente, o mediante ethernet. Esta última opción permite un enlace más rápido y entre mayores distancias así como la conexión con Internet a través de un router. La conexión por Ethernet se llevará a cabo siguiendo el protocolo TCP/IP, que es el usado en Internet y uno de los más populares para cualquier tipo de red.

Otra posibilidad que brinda el E.Reader es la utilización de dispositivos de almacenamiento USB como método para transportar los datos. Cuando conectamos un dispositivo de este tipo en el puerto USB del módulo, este automáticamente descarga en él todo los datos que tenga guardados en la memoria. Una vez hecho esto, el usuario puede visualizarlos utilizando el software apropiado. Obviamente este proceso solo puede hacerse de forma manual, lo que le resta practicidad.

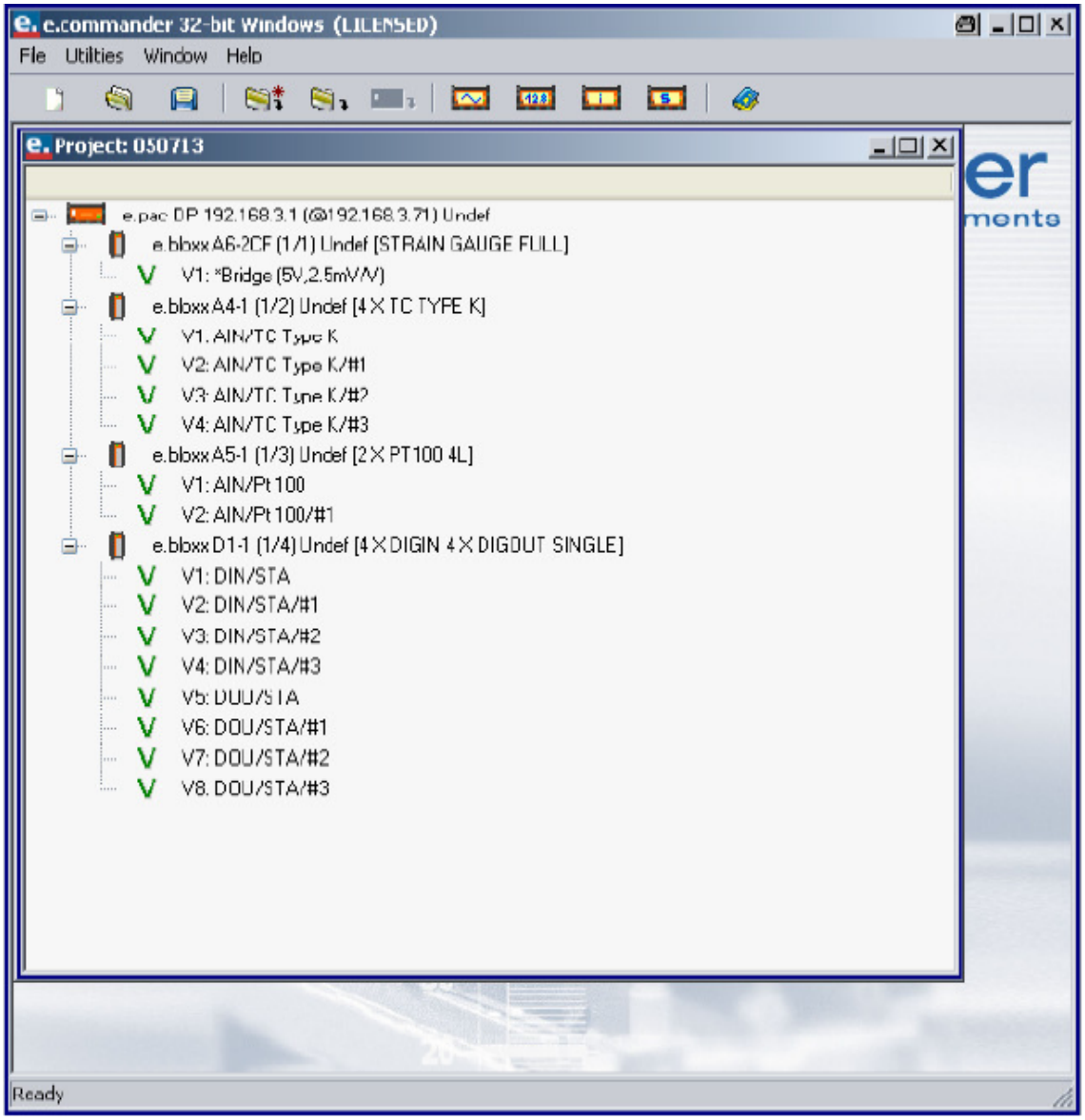
Desde Gantner, se hace hincapié en que el E.Reader se adecua perfectamente a instalaciones remotas y en condiciones duras. Las principales características que avalan esta afirmación son su bajo consumo de energía y su resistencia a calor y humedad. En cuanto a consumo, el E.Reader tiene la posibilidad de entrar en un modo de bajo consumos en el que simplemente se toman los datos y no se realiza ninguna otra operación con él hasta que pasa un intervalo de tiempo establecido, momento en el cual se realizan las operaciones requeridas. Una vez terminadas las acciones, el módulo vuelve al estado de letargo en el que solo toma datos. De esta manera puede reducirse el consumo hasta los 100mW, dependiendo del intervalo de tiempo de letargo y de operación. Tal como afirma la empresa, esta cantidad de consumo es ideal si se desea alimentar al E.Reader con dispositivos como paneles solares, que podría ser una solución adecuada en áreas remotas. Las condiciones ambientales de operación para el E.Reader van desde los 20 grados negativos hasta los 60 grados positivos y la humedad relativa que soporta puede variar entre un 0% y un 95%. Por lo que se puede ver el módulo es capaz de operar en condiciones de temperatura y humedad bastante extremas, lo que puede ser útil en localizaciones remotas.

El E.Reader necesita de un software para poder configurarlo y visualizar los datos recogidos. Gantner, ha creado una aplicación específica para sus productos, que permite manejarlos. Este Software es el E.Commander, y está disponible para descargarlo (en su versión de prueba) en la página Web de la compañía.

1.3 E.Commander

1.3.1 Introducción

Este Software es el que provee la compañía para acompañar a sus productos. Es la herramienta que permite la configuración y manejo de los módulos de Gantner. Está especialmente diseñada para entornos de pruebas, de ahí la gran cantidad de opciones que brinda y su aspecto industrial. Este es aspecto típico de la ventana principal del software, desde la que se accede a todas las funcionalidades del programa y que permite observar la configuración de los módulos conectados:

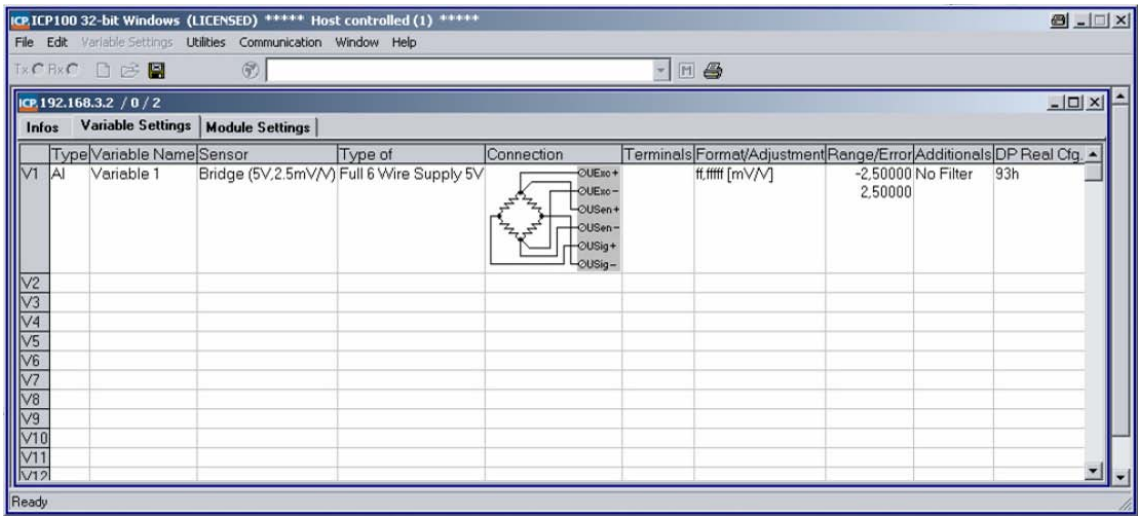


Como puede verse, en el centro de la ventana se encuentra el listado de módulos conectados al sistema. En este caso el módulo conectado es del tipo e.pac, que es otro de los modelos fabricados por gantner, y tiene asociados varios módulos del tipo

e.bloxx, de los que se pueden ver las entradas. La V de color verde a la izquierda de cada entrada indica que la entrada ha sido actualizada y por lo tanto puede ser utilizada.

1.3.2 Configuración

Una vez se tiene el sistema conectado apropiadamente, para ponerlo en funcionamiento, se han de añadir las especificaciones al software. Dado que para instalaciones grandes, esta operación puede ser costosa, también se ofrece la posibilidad de que el propio programa capture la configuración de la instalación. Una vez se haya hecho esta operación, se podrá visualizar una ventana similar a la que se ha mostrado unas líneas más arriba. Haciendo clic sobre el icono que representa al módulo puede accederse al menú de configuración del mismo. Desde este menú se podrán modificar todos los aspectos relacionados con la comunicación (como direcciones IP, contraseñas, protocolos o velocidades de transmisión) y con la captación de la señal proveniente de los módulos (frecuencias de muestreo, buffers de lectura, etc.). Si se hace clic sobre los iconos de las entradas también se accederá al menú de configuración de estas. Si se desea cambiar el sensor de una entrada, se accederá a una ventana similar a esta:



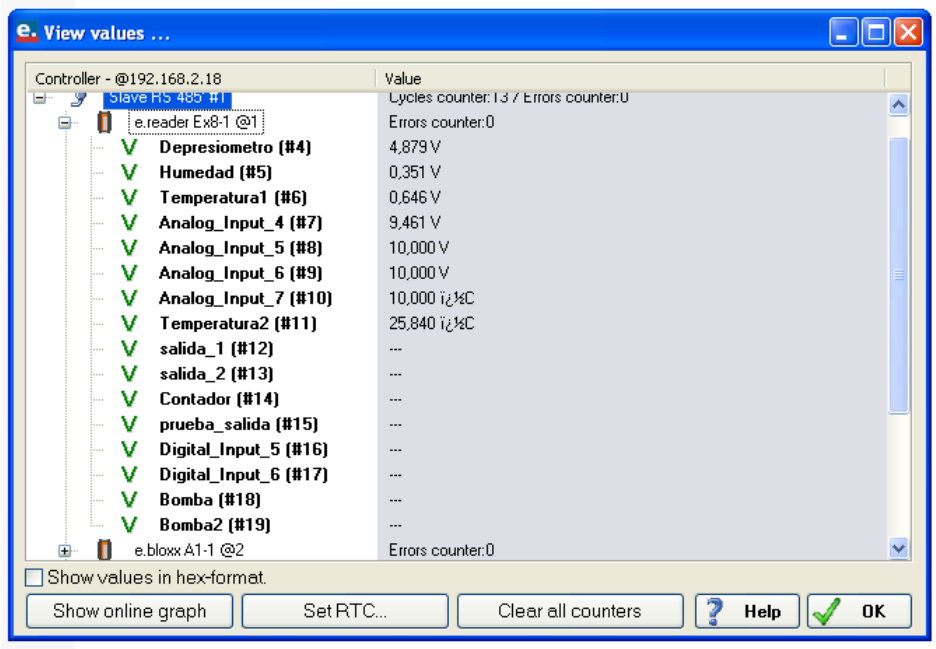
Aquí puede verse la configuración actual de una entrada típica. En el apartado Type se indica si se trata de una entrada o una salida y si es analógica o digital. En Variable Name se puede ver el nombre asociado a la entrada en cuestión. Sensor informa sobre el modelo de sensor que está conectado a la entrada. En este caso es un sensor de voltaje tipo puente con un rango de 5V y sensibilidad de 2,5mV. El modelo puede elegirse de una lista que se desplegará en una nueva ventana cuando se pulse en este apartado. En el apartado Type Of se dona más información sobre el módulo, como puede ser el número de terminales de los que dispone. Connection consta de un diagrama explicativo de cómo debe conectarse el sensor que se ha especificado en la entrada correspondiente. Esto es muy importante, ya que conectar un Terminal en el lugar inadecuado puede dañar tanto el módulo como el sensor. Estos son los apartados que deben modificarse a la hora de cambiar el módulo asociado a una entrada. El resto de los datos de la ventana también pueden modificarse pero no es obligatorio hacerlo. Format /Adjustment da la

precisión con la que se mostrarán los datos, Range muestra el rango en el que oscilarán los valores, etc.

Una vez hechos los cambios pertinentes en las entradas, se ha de proceder a la actualización de los módulos, es decir, a escribir los datos modificados en el E.reader a fin de que se puedan aplicar. Tras cambiar la configuración de las entradas se puede observar que aparece un icono en forma de asterisco rojo al lado del nombre de la entrada. Esto índica que la información de dicha entrada no ha sido escrita en el módulo, y por lo tanto no se aplicará. Pulsando los botones  ó  de la barra de herramientas comienza el proceso de actualización. La diferencia entre los dos botones es que el primero, el que tiene el asterisco rojo, solo actualiza las entradas que hayan sido modificadas. Sin embargo, el segundo actualiza el sistema completo, por lo que es más lento. Pueden usarse cualquiera de las dos formas indistintamente. En ambos casos el proceso de actualización tiene una duración bastante prolongada, de unos dos o tres minutos aproximadamente. Esto es debido a que se comprueba si la conexión está realizada correctamente y el estado de los módulos y sus correspondientes entradas. Cuando se completa este proceso aparecerán unos iconos en forma de V de color verde al lado de cada entrada. Esto indica que han sido actualizadas en los módulos de manera exitosa y están listas para usarse.

1.3.3 Captura de los valores

Una vez realizada la actualización de los módulos es posible comenzar con la captura de los valores. Si se quiere capturar los valores puntuales en tiempo real de la instalación, se deberá pulsar el botón  o seleccionar la opción Read Online Values From Concentrador del menú File. Esta opción, despliega una nueva ventana en la que se podrán observar todos los valores que entregan los sensores en el instante actual. Su aspecto es el siguiente:



Esta funcionalidad es interesante, sin embargo, resultaría aún más interesante que pudieran verse gráficos históricos que muestren la evolución de los valores que se seleccionen. Esta funcionalidad no está disponible en el e.commander, aunque sí que cuenta con una pequeña función para representar gráficamente un valor. Esta función se pone en marcha pulsando el botón  o seleccionando la opción Read Online Buffer From Concentrator. De esta manera se podrán visualizar los datos del intervalo de tiempo guardado en el buffer del E.reader. No obstante, los datos no se almacenan, por lo que si se deseara volver a visualizar los datos no sería posible a no ser que se guarden manualmente. Este sistema es a todas luces muy poco práctico. La opción que propone la empresa para visualizar los datos es utilizar un software adicional también diseñado por ellos llamado test.viewer o un paquete de freeware llamado Green Eye Reader. No se ha estudiado la funcionalidad de estos programas debido a que el simple hecho de estar obligado a utilizar dos programas distintos ya resulta poco práctico.

Por lo tanto, el software que proporciona Gantner para el uso de sus módulos no se ajusta a las necesidades de este proyecto. Se tratará de crear una aplicación más sencilla, intuitiva y que sea capaz de presentar los datos de una manera atractiva y clara.

1.4 La aplicación

Como ya se ha dicho, se pretende que la interfaz se adecue a la actividad de una explotación agraria. Una explotación agraria es aquella cuya producción son los productos agropecuarios, entre los que pueden distinguirse dos clases, los productos agrícolas y los ganaderos. Dependiendo del tipo de producción de la explotación en cuestión, parece que las necesidades en cuanto a parámetros a controlar serán distintas. Sin embargo, esto no es del todo cierto, ya que los parámetros más importantes como lo son la temperatura, la humedad relativa, la luminosidad o la concentración de CO2 en el aire son comunes a cultivos y cría de animales.

Como ejemplo, pueden compararse los casos de una granja avícola y una explotación agrícola. En ambos casos la temperatura es un factor muy importante ya que las aves han de tener una temperatura estable para facilitar la reproducción o la incubación de huevos y los cultivos también la necesitan para desarrollarse. La humedad relativa quizá es más crítica en el caso de los cultivos, sin embargo, también influye en la vida de los animales ya que una humedad no óptima puede acarrearles enfermedades. La luminosidad es un factor vital tanto para los cultivos como para la cría de las aves, ya que en el primer caso es lo que desencadena la fotosíntesis y en el segundo regula los tiempos de alimentación o sueño de los animales. Por último, es muy importante conocer el nivel de CO2, en caso de cultivos debido a que es igual de necesario para el desarrollo de los mismos que el oxígeno, y en el caso de animales, una concentración demasiado elevada de este gas puede producir la muerte del ganado o graves consecuencias en él.

Como se puede ver, estos cuatro parámetros son de vital importancia para todo tipo de explotaciones, por lo que es conveniente contar con sensores para la medición de dichas variables. Uno de los requisitos indispensables de esta interfaz será la flexibilidad, por lo que no hará falta ninguna rutina específica para mostrar estas variables, sin embargo, merece la pena conocer el tipo de datos que se deberán representar a fin de estructurar de la manera adecuada la interfaz de usuario.

En una explotación agraria, no tiene porqué haber personal técnico, por lo que otro de los requisitos de la interfaz será la capacidad de representar los datos de una manera intuitiva, que pueda ser comprendida por un usuario medio. Se deberá prestar especial atención a la sección de configuración de los módulos, manteniéndola simple dentro de lo posible y sin perder flexibilidad y capacidad de configuración. Esta sección será posiblemente la menos intuitiva de cuantas consta el programa ya que habrá por fuerza ciertos parámetros técnicos, para aliviar esta complejidad, se creará una pequeña guía de uso en la que se explicará la relevancia de cada dato configurable o visualizable.

Por último, en este tipo de explotaciones, es necesario tener una visión histórica de los parámetros que se han comentado, ya que de esta manera se podrán reconocer patrones que afectan a la calidad de la producción final, con lo que repitiendo los patrones positivos y evitando los negativos se ayudará a optimizar la explotación. Para que la interfaz sea adecuada a esta aplicación, se han de cumplir todos estos requisitos.

Como se ha visto en la sección anterior, el software E.Commander de Gantner, ofrece mucha flexibilidad en la configuración del equipo. Las opciones que nos brinda son numerosas, sin embargo, su manejo no es nada intuitivo. La configuración de los parámetros es muy compleja, ya que requiere que se le especifiquen todas las características del sensor y nos muestra opciones atípicas o con nombres poco inteligibles para un usuario medio. Además, la obtención de los datos requiere de algo de tiempo de espera y configurar los tiempos de captura requiere de varios pasos. La visualización no tiene un formato muy atractivo y no es posible crear representaciones gráficas de los datos recogidos en un periodo de tiempo solo con el E.commander, sino que se debe utilizar otra aplicación. Añadido a todos estos factores está el hecho de que la interfaz solo está disponible en inglés, francés o alemán.

El fin de este proyecto, es el de crear una nueva interfaz en la que se corrijan las deficiencias del software de Gantner y se mantengan los aspectos positivos del mismo. Existen algunos programas con funcionalidades similares al que se pretende crear, como lo son el propio E.commander y VinWizard de Wine Technology Marlborough, este último, un software específico para bodegas vinícolas. Tomando a ambos como base se diseñará la nueva interfaz. Para comenzar, habrá que estudiar el E.commander y su interacción con los módulos a fin de imitarlas. La principal herramienta que se usarán para ello, será el programa Wireshark, un analizador de protocolos que ayudará a observar la comunicación que se lleve a cabo entre el software y los módulos.

2 Primeros Pasos

2.1 Estudio de la comunicación

2.1.1 Introducción

Como se ha explicado antes, el paso más importante del inicio del proyecto es estudiar la comunicación entre el E.commander y el E.Reader, ya que si se comprende la manera en la que se hace la comunicación, luego simplemente habrá que replicarla. Se sabe que el módulo envía los datos captados por los sensores al software y este último envía la configuración de las entradas y salidas. Para observar esta comunicación será necesario utilizar un software analizador de protocolos, en este caso el elegido a sido Wireshark, ya que se tenían conocimientos previos del funcionamiento del mismo.

2.1.2 Wireshark

Este programa permite visualizar todo el tráfico de paquetes de una red, pudiendo capturar los paquetes deseados para su análisis en profundidad. Para que la captura de paquetes sea más efectiva, el programa dispone de filtros, por lo que se podrán obviar archivos según el tipo de protocolo al que pertenezcan, por ejemplo podrían ignorarse los paquetes de datos de protocolo ICMP (como son las peticiones y respuestas de eco) o visualizar únicamente los asociados al protocolo FTP. Además de esto, el programa también ayuda a la interpretación de los paquetes capturados, ya que detecta el protocolo al que pertenecen y separa las informaciones de la cabecera y los datos.

Configurar Wireshark es muy sencillo, simplemente se ha de especificar el puerto en el que debe escuchar y configurar el filtro de captura. El programa establece automáticamente lo que se conoce como modo promiscuo, por el cual es capaz de recibir todos los paquetes que circulan por una red de datos independientemente del destinatario real de los mismos. Una vez capturados los paquetes de información, Wireshark permite leerlos y analizarlos con mucha facilidad ya que es capaz de reconocer más de 480 protocolos. Los datos son mostrados en formato hexadecimal y convertidos a ASCII para facilitar la lectura de datos de tipo texto, aunque esta función puede no ser de utilidad si los datos recibidos no son de esta clase. Con los datos en hexadecimal y llevando a cabo las conversiones pertinentes puede descifrarse cualquier mensaje que se incluya en los paquetes capturados.

En este caso, la red a la que se conectará Wireshark será la 192.168.2.0/24. Para quien no esté familiarizado con la nomenclatura de las redes, esta comprende las IPs entre la 192.168.2.0 y la 192.168.2.255. El número final, que se encuentra después de la barra es el que índica la longitud de la máscara de subred, que en esta caso será de 24 bits. La máscara de subred se usa para determinar el número de IPs distintas que podrá contener la red, ya que sirve para separar la parte de la dirección que corresponde al identificador de red de la parte asignada al host. En otras palabras, la longitud de la máscara de subred corresponde también al número de bits (de la dirección) que se usan para

nombrar la red. Exististe otra forma de especificar esta máscara, que puede aclarar la forma en la que la separación se lleva a cabo: 255.255.255.0. Si se traduce el número anterior a bits se obtendría la cadena siguiente: 11111111 11111111 11111111 00000000. Si se traduce también una dirección IP (por ejemplo la 192.168.2.18) se obtendrá esta otra cadena: 11000000 10101000 00000010 00010010. Si se combinan multiplicándolas Bit a Bit, se puede observar que la parte de la IP reservada al host se anula, quedando solo visible la dirección de la red, el resultado sería: 11000000 10101000 00000010 00000000 ó 192.168.2.0.

Para comenzar el estudio de la comunicación, en primer lugar se ha de configurar el puerto que se conectará al E.reader con una IP contenida en la de la red a la que pertenece. Dado que el e.reader viene configurado por defecto con la IP 192.168.2.18/24 se ha elegido la IP 192.168.2.15/24. Se pone en funcionamiento Wireshark y se activa la captura. En este punto no se obtiene ningún resultado ya que la comunicación no ha empezado aún. Se inicializa el E.commander y se activa la función *Read Online Values*, que es la encargada de capturar los valores en tiempo real. En este momento comienzan a aparecer los paquetes de datos, una captura de 10 ó 15 segundos será suficiente para observar las características principales de la comunicación. Una vez transcurrido este tiempo puede detenerse la captura y la transmisión de datos.

2.1.3 Primeros paquetes de información

Tras un primer estudio superficial de los paquetes capturados, parece claro que la comunicación se divide en dos o tres partes. La primera parte, son una serie de ecos entre el módulo y el software. Esta primera parte es de longitud variable, ya que en sucesivas capturas se encontraron desde uno hasta siete ecos. Cada eco está formado por dos paquetes, el primero es enviado por el software y en el se pide una respuesta. El segundo paquete es la respuesta que emite el módulo. Estos paquetes no contienen información alguna, simplemente se utilizan para comprobar que la conexión entre equipos es posible. El protocolo al que pertenecen este tipo de envíos es el ICMP (Internet Control Message Protocol). Estos ecos se conocen con el nombre de *ping* y son comúnmente utilizados en el ámbito de la informática. Además del ping, el protocolo ICMP contiene otra herramienta de uso extendido conocida como traceroute o tracert. Con ella se puede controlar el trayecto de un paquete de datos desde el origen hasta el destino, por lo que es muy útil en diagnósticos de redes. Es muy probable que no sea necesario implementar esta parte en la interfaz, ya que aunque se hace a fin de asegurar que la conexión es correcta, si se suprimiera el resultado sería el mismo previsiblemente.

2.2 Transmisión FTP

2.2.1 Conexión

La segunda parte se hace bajo las directrices del protocolo FTP (nivel aplicación en OSI). Es la parte más compleja y también la única que tiene una longitud (número de paquetes) fija. En primer lugar se puede observar la conexión entre los dos equipos, que se lleva a cabo en tres pasos, como ocurre en los protocolos basados en TCP. Este procedimiento se denomina *Three Way Handshake*. En el primer paso, el host (software) envía un paquete SYN, con el que informa al servidor (módulo) que desea iniciar una conexión con él. Al recibirlo, el servidor responde a dicho mensaje con un paquete SYN/ACK, que confirma que se ha recibido la petición y que esta es válida. En el último paso, el host emite un último paquete ACK (acknowledgement), para confirmar que la conexión ha sido establecida con éxito. En este caso al three way handshake se le añade la conexión al servidor FTP mediante nombre de usuario y contraseña. Esto añade seguridad a la conexión, sin embargo esta es limitada ya que la transmisión no está cifrada, por lo que pueden visualizarse todos los datos con un simple sniffer como Wireshark. Al terminar el three way handshake, el módulo emite un paquete en el que informa de que el servidor FTP está listo. Este es el primer paquete que contiene información, por lo que, como todos los de este tipo lleva el indicador PSH (Push). En el siguiente paquete se proporciona el nombre de usuario. Dada la importancia de la información transmitida, en la cabecera del paquete se encuentra activo el indicador No Fragmentar.

Para facilitar la transmisión de datos a través de Internet es posible descomponer los paquetes de datos más grandes en varias partes. De esta manera, si una de las partes resulta extraviada, no se perderá mucha información y se podrá volver a enviar rápidamente. La fragmentación se lleva a cabo en los enrutadores correspondientes a cada red ya que distintas redes pueden tener distintos tamaños máximos para los paquetes que se transmitirán por la red. Esta manera, es la más eficiente para el envío de datos por una red de las características de Internet. El único inconveniente de este sistema, es el de tener que recomponer el paquete original (no fragmentado) en el destino. Para ello, los paquetes están numerados, y para evitar que si se pierde alguna de las partes la información quede incompleta se utiliza el protocolo de ventana deslizante, del cual existe numerosa información sobre su funcionamiento en Internet. En caso de información importante, como en esta ocasión, existe la posibilidad de indicar a los enrutadores que el paquete en cuestión no debe ser fragmentado. Si un paquete que no debe ser fragmentado llega a un tramo de red que tiene una longitud de datagrama máxima menor a su tamaño, el paquete no será transmitido y se perderá. En las nuevas versiones del protocolo IP la fragmentación de paquetes en enrutadores remotos ya no será necesaria, ya que mediante mensajes ICMP, podrá elegirse el tamaño de los paquetes de tal manera que podrán ser transmitidos por todos los segmentos de la ruta sin inconvenientes.

El módulo responde al envío del nombre de usuario confirmándolo y requiriendo la contraseña, por lo que el software la envía. Si esta es correcta se envía un mensaje de confirmación y el módulo queda en espera de nuevas comunicaciones.

2.2.2 Transmisión de información sobre el sistema

Tras esto, el software comienza una recopilación de información sobre todas las entradas y variables del módulo. En primer lugar, solicita información general sobre el módulo como son marca, modelo y versión mediante el comando SYST. Tras esto, pide un listado de todos los archivos que se encuentran en el servidor. El módulo envía el listado, fragmentado en varios paquetes, a los que él software va dando confirmación de que ha recibido. Antes de la transferencia del archivo del listado, o de cualquier otro archivo como se verá más adelante, se puede observar que el software envía un paquete con un comando PORT. Este comando sirve para configurar los sockets (más específicamente los puertos) para la comunicación y debe ser enviado antes de cualquier transferencia. Los puertos a configurar dependen del modo en el que se encuentre el servidor.

Existen dos modos: Activo o pasivo. En el modo activo, el servidor de datos crea las conexiones entre su puerto 20 y un puerto aleatorio del cliente. Esto genera un problema de seguridad ya que el cliente debe estar dispuesto a recibir conexiones aleatorias en sus puertos, lo que deja vía libre a extraños que quieran acceder al equipo. De hecho, la mayoría de los cortafuegos bloquean automáticamente este tipo de conexiones. Para evitar esto existe el modo Pasivo. En este modo es el cliente el que especifica el puerto al que ha de conectarse el servidor, solucionando los problemas anteriores.

Además del comando PORT, previamente al envío de archivos se configura también el tipo de dato mediante el comando TYPE. Existen dos tipos: El A (modo ASCII) y el I (modo binario). En el modo ASCII los datos se convierten a su representación en este tipo de datos. Se utiliza cuando los datos que se van a recibir o transmitir son tipo texto. El modo binario se utiliza en el resto de casos, ya que los datos se envían byte por byte y las conversiones necesarias (si es que hay alguna) se harán en el receptor. En este caso, antes de pedir el listado, se activa el modo ASCII, ya que los datos del listado son de tipo texto. Sin embargo, antes de la transmisión de los archivos que se especificarán más tarde, se activa el modo binario, ya que no se conoce el contenido de dichos archivos.

2.2.2.1 El archivo #actual.sta

Tras finalizar la transferencia del listado, se vuelve a repetir la operación y se pide un nuevo listado, a los que el servidor responde de igual manera. Una vez terminado él segundo envío, se activa el modo binario. Esto es señal de que los datos a transmitir no van a ser de tipo texto, como se ha explicado antes. Se configura un nuevo puerto, como en las dos ocasiones anteriores y se realiza la petición de un archivo llamado *#actual.sta*. Este es uno de los archivos del listado. La petición se hace utilizando el comando RETR, que es el encargado de dar la orden al servidor de que envíe el archivo especificado al cliente.

El archivo #actual.sta contiene información sobre el sistema, como versión y fecha de la aplicación, y sobre el módulo, como marca y modelo. Además, en este archivo se puede consultar el estado en el que se encuentran los módulos del sistema. En primer lugar estados generales, donde se puede ver el modo en el que está configurado el e.reader (modo lectura o modo configuración), en este caso modo configuración, y se puede comprobar si la configuración de la instalación es correcta, en cuyo caso este apartado constará como estable. También se muestra información sobre el estado de la ejecución. En este caso, se muestra que el servidor FTP está activo, lo cual es lógico ya que se están descargando archivos de él. Consta también un espacio para posibles mensajes de error, que en esta ocasión está vacío. Tras esta información general, se encuentra un listado con el estado de cada uno de los módulos de la instalación. La lista contiene 10 elementos. El primero, de tipo MK21#, es el e.reader, y su estatus es correcto (OK en la nomenclatura que se utiliza en el archivo). Los siguientes ocho elementos de la lista, marcados como MK18#, son loas módulos de entrada del e.bloxx y su estatus es correcto también. El último elemento, nombrado como MK33#, es el módulo de salidas digitales del e.bloxx y se encuentra en estatus OK. Además del estado, también se especifica para cada entrada de la lista la versión del módulo y la aplicación a la que pertenecen, y añadido a esto, una numeración marcada como SNR.

Una vez finalizada la transferencia del archivo #actual.sta, se vuelve a pedir el listado dos veces, como se hizo anteriormente. Tras esto, se vuelve a activar el modo binario y se realiza la petición de el archivo #summary.sta. Este archivo también se encuentra en el listado del servidor y por lo tanto se acepta la descarga.

2.2.2.2 El archivo #summary.sta

El archivo #summary.sta contiene una descripción detallada de los parámetros de configuración de todas las variables del sistema, desde variables virtuales como contadores hasta las entradas. Dado que en el archivo está contenida información sobre entradas de distintos módulos, se han de separar de algún modo. La solución que ha tomado la empresa ha sido la de numerar cada entrada con dos letras y dos números como veremos a continuación. Al haber también variables que no pertenecen a ningún módulo, se ha optado por numerarlas de otra manera, utilizando solamente una letra y un número. El archivo comienza con un apartado llamado GENERAL, que tiene la siguiente forma:

```
[GENERAL]
Type=MK172006#
Location=UPNA
SNR=640141
AppName=e.reader (V3)
AppVersion=V0.28 2008-09-23
Vendor=GANTNER instruments
SampleRate=1.000000e+00
VariablesCount=4
SlavesCount=4
```

Como se puede ver, en este apartado se proporciona información general del módulo una vez más, sin embargo esta vez se añaden tres datos interesantes: La frecuencia de muestreo (SampleRate), el número de variables virtuales (VariablesCount) y el número

de dispositivos en la red (SlavesCount). Se intentará explicar porqué el número de dispositivos que consta en SlavesCount en principio no concuerda con el número real, ya que solo se dispone de un E.reader, un e.bloxx y un e.bloxx de salidas digitales, y SlavesCount es igual a 4. También es importante resaltar que la frecuencia de muestreo es de 1 Hertzio, es decir, una muestra por segundo. El resto de parámetros son comunes a las anteriores veces que se pedía información general sobre el módulo.

El siguiente apartado tiene de nombre FEATURES (características en ingles) y contiene dos sentencias a tener en cuenta al leer el buffer. En este caso no serán necesarias. Tras esto, comienza la descripción de las variables virtuales, llamadas de esta manera debido a que no están asociadas a una entrada física. En esta ocasión se incluyen cuatro numeradas V0, V1, V2 y V3. V0 y V1 tienen configuraciones idénticas, así como V2 y V3 entre ellas. Este es el formato:

```
[V0]
Name=CycleCounter
VarType=ARI
DataDirection=I
DataType=USINT32
Format=%8.0f
Unit=
AccessIndex=0
InpSplitDataFieldOffs=0x0000
InpCombDataFieldOffs=0x0000
```

En primer lugar se encuentran el nombre, el tipo de variable y si es una variable de salida o de entrada. En este caso es una variable de tipo entrada (I). DataType especifica el formato del dato, en este caso un número entero sin signo de 32 bits (usint32) y Format da el número de bits después de la coma en caso de que el formato sea de coma flotante. Dado que esta variable corresponde a un contador (como se puede deducir por el nombre de la misma), Unit no presenta ningún valor. AccessIndex informa de que esta variable es la número 0, es decir la primera, y los dos siguientes parámetros dan información de la posición en la que se encuentra escrita dicha variable en caso de que fuera necesario extraer los valores. Dado que es la primera variable, la posición es 0. V2 y V3 son similares pero cambia el tipo de dato, que es coma flotante (FLOAT) y Format, que indica que hay tres bits después de la coma. Obviamente también cambian los tres últimos parámetros, dependiendo a que variable nos refiramos.

Tras finalizar con la descripción de estas variables, comienza con los módulos físicos. En primer lugar da información del módulo y después sobre las entradas asociada al mismo:

```
[M0]
Type=MK21#
UartIndex=0
Address=1
NotSynchronizedSampleRate=9.900990e-01
VariablesCount=16
Location=eReader_Default
```

Como se puede ver la numeración cambia con respecto a la anterior. Para diferenciarlos de las variables se cambia la V (probablemente referida a Variable) por una M (de

Módulo). El tipo (MK21#) es el mismo que se vio anteriormente en el archivo #actual.sta, y coincide con el asignado al E.reader. El dato más relevante de este apartado es sin duda VariablesCount, que indica el número de entradas configuradas en el dispositivo. Las entradas contenidas en este módulo se representan de la siguiente manera:

```
[M0_V0]
Name=Depresiometro
VarType=AIN
DataDirection=I
DataType=FLOAT
Format=%8.2f
RangeMin=-10
RangeMax=10
Unit=V
AccessIndex=4
InpSplitDataFieldOffs=0x0010
InpCombDataFieldOffs=0x0010
```

A la numeración anterior se le añade el indicador V más el número asignado a la entrada empezando desde cero. Después del nombre, se encuentran el tipo de variable, que en este caso es una entrada analógica (AIN es Analog Input), y DataDirecction, que puede ser I u O, dependiendo de si la variable está reconocida como de tipo entrada o salida. Tras esto, se encuentra el tipo de dato, que habitualmente es de coma flotante (también se encuentran valores como USINT32 SINT16 ó BOOL), y Format, que como se ha explicado informa del número de dígitos después de la coma. En este caso serán dos, como se puede apreciar. Otra información importante es el rango de valores de la entrada, dado aquí por los límites (mínimo y máximo) entre los que oscilarán los valores. A diferencia de cuando se hablaba de las variables virtuales, aquí si que se muestra un valor en Unit. Los valores que puede tomar este parámetro coinciden con los que el módulo es capaz de medir (Voltios, Amperios y Resistencia). Como en los casos anteriores, los tres últimos parámetros sirven para identificar la entrada. Esta estructura se repite para el resto de entradas asociadas al módulo. Como caso especial, una de las entradas está configurada para medición de temperaturas. Los sensores encargados de este cometido son de resistencia variable y por lo tanto en el apartado de unidades debería aparecer algún distintivo que lo indicara, sin embargo consta tan solo el símbolo de grados Celsius (ºC). Otros casos especiales son las entradas y salidas digitales, que hacen que VarType tome los valores DIN y DOU respectivamente. El siguiente módulo es el e.bloxx y su configuración general es la siguiente:

```
[M1]
Type=MK18#
UartIndex=0
Address=2
NotSynchronizedSampleRate=9.900990e-01
VariablesCount=3
Location=Undef
```

Su configuración es muy parecida a las del E.reader, pero en este solo consta de tres entradas configuradas. Una de ellas es una salida digital:

```
[M1_V2]
Name=salida_digital
VarType=DOU
DataDirection=IO
DataType=FLOAT
Format=%8.0f
Unit=
AccessIndex=22
InpSplitDataFieldOffs=0x003F
OutSplitDataFieldOffs=0x0007
InpCombDataFieldOffs=0x003F
OutCombDataFieldOffs=0x003F
```

Como se puede observar la estructura es muy similar a la de las entradas como la que se ha estudiado anteriormente. Una de las diferencias más notables es la ausencia de datos en las unidades, lo cual es comprensible ya que habitualmente este tipo de salidas se utilizan para implementar alarmas o activar actuadotes, por lo que las unidades no son necesarias. La otra diferencia es la adición de dos parámetros entre los que se utilizaban para situar la entrada. La funcionalidad exacta de estos nuevos parámetros no está del todo clara, pero no parecen ser de vital importancia para el desarrollo de la futura aplicación.

El siguiente módulo también consta como de tipo e.bloxx, pero en este caso solo consta de una entrada configurada en él. Parece que por alguna razón el módulo único e.bloxx, ha sido seccionado en dos o bien por el software o bien por el E.reader. Si la causa es el software puede deberse a alguna configuración incorrecta, sin embargo si la causa es el E.reader puede que haya habido algún problema en el reconocimiento del módulo. Dado que el dispositivo funciona correctamente y este problema no parece extenderse a otros ámbitos, se podrá pasar por alto. La entrada perteneciente a este módulo está configurada para la medición de temperaturas.

El último módulo representado en el archivo es el de las salidas digitales. Viene identificado como tipo MK33# y cuenta con 4 entradas configuradas, todas ellas de tipo Booleano. De esta manera termina el archivo #summary.sta, en el que se ha podido ver la configuración básica de las entradas. En principio, este archivo parece ser el adecuado para ejecutar cambios de configuración en el módulo, sin embargo, como se verá más adelante esto no es posible debido a que el módulo no acepta cambios en el archivo. Esto es debido a que #summary.sta no es más que un resumen que crea el E.reader con la configuración principal de las entradas.

2.2.3 Desconexión FTP

Una vez finalizada la transferencia del archivo #summary.sta el software ya está listo para comenzar la transmisión de datos, por lo que pide la desconexión del servidor mediante el comando Quit. El servidor responde afirmativamente a la petición y desconecta el enlace FTP, sin embargo, la conexión TCP sigue abierta. Esto se soluciona inmediatamente ya que el software envía un paquete con el indicador FIN (que es el que indica que se quiere cerrar la conexión). Así como en la conexión se observaba un three way Handshake, la desconexión también tiene su protocolo, que en este caso difiere del utilizado en TCP. En primer lugar, como se ha visto, se pide la desconexión. El servidor responde a la petición con un paquete con los indicadores FIN

y ACK, que significan que se acepta la desconexión. El software al recibir este paquete envía un último paquete ACK y finaliza la escucha en el puerto. Aunque el cliente ya haya cerrado la conexión, el servidor no la finaliza hasta que no recibe este último ACK y envía uno propio de respuesta. Obviamente este último paquete de confirmación por parte del servidor nunca llegará a ser recibido ya que el cliente había cerrado la conexión previamente. Todo este proceso se hace previamente a iniciar la transferencia de datos. El último paquete perteneciente a esta parte, se recibe 3.78 segundos más tarde que el primero, por lo que la pérdida de tiempo debido a estas transferencias es pequeña. De hecho, si se tiene en cuenta que solo la parte de la conexión al servidor dura 2.45 segundos, se puede deducir que el tiempo para descargar los archivos y recibir los listados es despreciable.

2.3 Transmisión de los datos TCP

2.3.1 Introducción a TCP

La tercera parte de la comunicación se hace bajo las normas del protocolo TCP y es la en la que se transmitirán los datos, es decir, los valores capturados por los sensores de la instalación. El protocolo TCP es un protocolo de comunicación orientado a conexión encuadrado en el nivel de trasporte del modelo OSI. TCP es el encargado de empaquetar por primera vez la cadena de bytes entregada por la aplicación para que los datos puedan ser enviados de forma fiable a su destino. El protocolo incluye herramientas de control de flujo y de congestión para evitar la saturación del receptor. Al ser orientado a conexión, es necesaria una negociación entre los equipos previamente al envío de datos. Como se vio en el caso del protocolo FTP, la conexión se realiza mediante el sistema conocido como Three Way Handshake. Este protocolo es el que permite que puedan crearse varias conexiones distintas por el mismo enlace, ya que es en el que se especifica el concepto de socket, que es la unión entre un número de puerto y una dirección IP. De esta manera, y teniendo en cuenta que cada interfaz de conexión (conector) solo puede tener una dirección IP, pueden tenerse tantas conexiones activas como puertos hay (teóricamente un máximo de 65535).

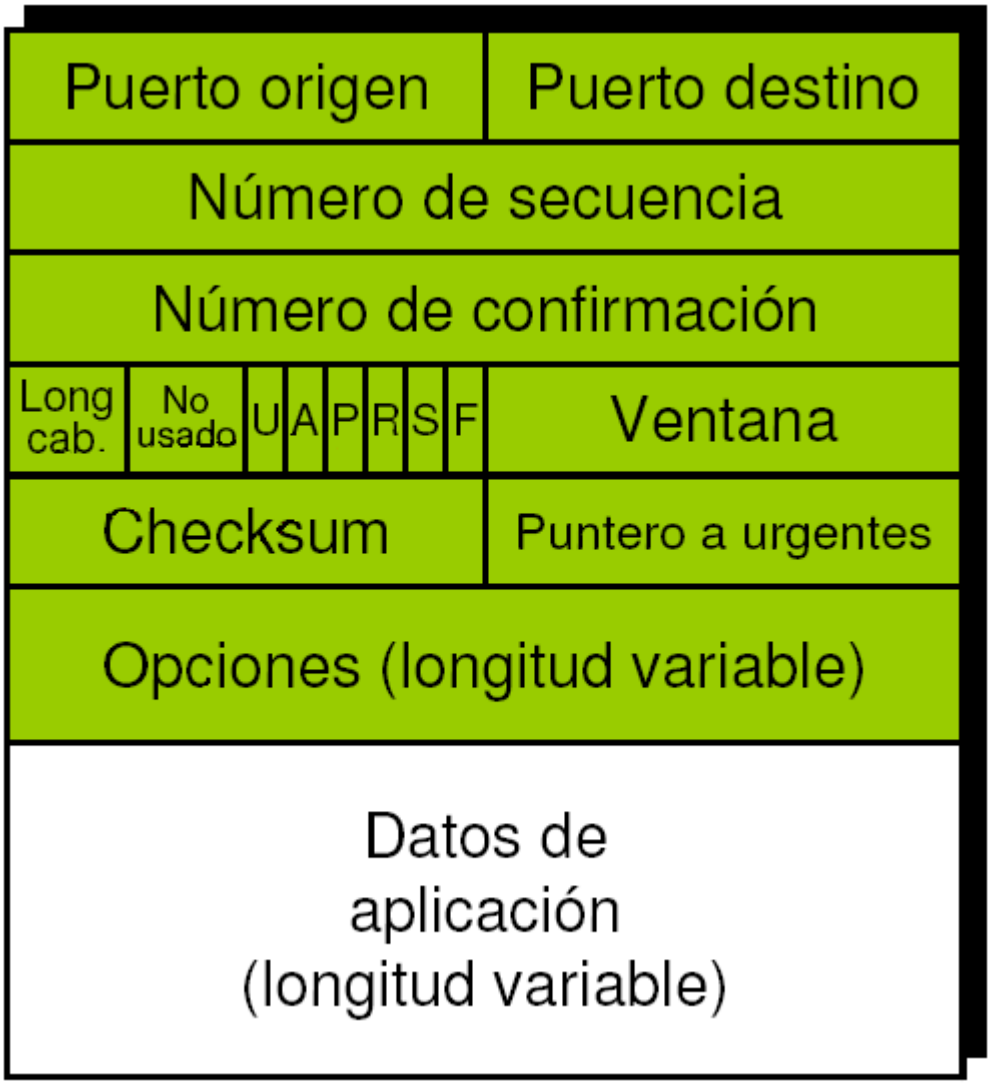
Existen algunas convenciones en cuanto a la elección de puertos ya que están divididos en tres tipos: Los puertos con número inferior a 1024 son puertos reservados y son usados por protocolos conocidos (20 y 21 para FTP, 23 para Telnet, 80 para HTTP, etc.). Los puertos comprendidos entre el 1024 y el 49151 son puertos registrados y pueden ser usados por cualquier aplicación. Existen listas en las que se especifica cual es el protocolo de cada puerto, como la de IANA, la organización que supervisa la numeración y nomenclatura en Internet. En las comunicaciones tipo cliente-servidor tanto los puertos reservados como los registrados son utilizados por los servidores. Por último, el rango de 49151 hasta 65535 corresponde a puertos dinámicos, que son los utilizados habitualmente por el cliente. Que el servidor utilice los puertos reservados o registrados tiene mucho sentido, ya que es importante ser accesible y que el cliente conozca el protocolo con el que ha de comunicarse.

El control de flujo es posible gracias al envío de paquetes de acuse de recibo o ACK en conjunción con el protocolo de ventana deslizante. Este protocolo permite evitar congestiones en el receptor y asegurar que la información ha sido recibida correctamente. El funcionamiento del protocolo es sencillo: El emisor y el receptor cuentan con sendos buffers y deben tener un tamaño de ventana máximo (por ejemplo 5). El emisor comienza a enviar paquetes hasta un número total de 5 (como el tamaño de la ventana) y espera confirmación. Cuando se recibe el primer paquete el receptor envía un ACK en el que consta el identificador de dicho paquete. Cuando el emisor ve que uno de los paquetes ha sido recibido envía los siguientes 5 paquetes después del que ha sido confirmado (en este ejemplo serían los paquetes 2, 3, 4, 5 y 6). Si por alguna razón el paquete 2 se perdiera y no llegara al receptor, este enviaría de nuevo el ACK correspondiente al paquete 1, aunque ya contara con los paquetes siguientes. De esta manera, el receptor no confirma la llegada de un paquete hasta que no cuenta con todos

los anteriores, lo que asegura la recepción íntegra de la información. Existen en Internet ejemplos gráficos de este protocolo que pueden ser consultados en caso de necesidad.

2.3.2 El segmento TCP: La Cabecera

Como se ha desvelado antes, el protocolo TCP es el encargado de empaquetar el flujo de bytes proveniente de la aplicación. Existen una serie de normas para el empaquetado de dicha información de tal manera que sea fácilmente extraíble y se incluya en el paquete toda la información adicional necesaria. El paquete TCP (también conocido como segmento) tiene la siguiente estructura:



Como se puede ver, se añade una cabecera a los datos proporcionados por la aplicación, que se encuentran al final del segmento. El primer apartado de la cabecera, con una longitud de 16 bits es el número de puerto desde el que se envía el paquete. El segundo, con una longitud igual a la anterior, es el número de puerto del dispositivo al que se envía el paquete. Estos dos datos, junto con las direcciones IP, son imprescindibles para el correcto direccionamiento del paquete. El tercer apartado, con una longitud de 32bits,

índica que byte de datos es el primero que se puede encontrar en el paquete. Se usa para comprobar que todos los datos han sido transmitidos. El número de confirmación (también de 32bits) se utiliza en caso de que esté activo el indicador de ACK, y en él se podrá encontrar el número de secuencia que se espera recibir. Este campo es importante para el funcionamiento del protocolo de ventana deslizante. El siguiente apartado da la longitud de la cabecera del segmento en palabras de 32 bits. La longitud de la cabecera puede tener desde un mínimo de 5 palabras de 32 bits (20 bytes) hasta un máximo de 15(60 bytes). Los siguientes cuatro bits conforman un campo no utilizado reservado para aplicaciones futuras, por lo tanto deberían ser cero en todos los casos. Los siguientes ocho apartados (aunque por simplicidad en el diagrama solo aparecen seis) tienen una longitud de un bit cada uno y son conocidos como Flags.

El primer Flag se conoce como CWR y se utiliza para evitar congestiones en la red. El segundo es el ECE, indica que el receptor es capaz de realizar un tipo de operación. Este también es utilizado para controlar las congestiones en la red. El tercer bit se conoce como URG, y se utiliza para notificar que el campo Puntero a urgentes es importante. Esto se hace así debido a que en la mayoría de casos este campo puede ser ignorado. El siguiente Flag es el ACK. Como se ha visto, se utiliza para confirmar la recepción de paquetes. Además de para esto, también indica si es necesario leer el campo de Número de confirmación. El siguiente bit corresponde al Flag PSH, más conocido como Push, que se activa cuando se envían datos. El sexto apartado es el RST, que se utiliza en caso de cierre abrupto de la comunicación. Si está activo se informa al receptor de que la desconexión es inmediata, por lo que no atenderá a ninguna respuesta. El siguiente bit corresponde al SYN, que como se ha visto se utiliza al iniciar una conexión. El último Flag es el de FIN, del que también se ha observado el uso en la finalización de transmisiones. Los Flags son la manera más directa de entender una comunicación, ya que nos indican en que fase de la misma nos encontramos, ya sea la conexión (que se reconocerá por el uso del SYN), la desconexión (en la que se usará el FIN) o la transmisión de datos (que será un flujo constante de PSH y ACK).

Los siguientes 16 bits corresponden a un campo conocido como ventana, en el que se especifica la longitud de la ventana del receptor, es decir, la cantidad de bytes que se espera recibir. El campo Checksum, también de 16 bits, es la suma de verificación, con la que se pueden detectar errores en la cabecera o el payload (los datos). En este campo, se da el valor que ha de tener la suma de ciertos bits del paquete, cuando se transmite el paquete el receptor hace la suma y comprueba que el resultado es el mismo que el que consta en este campo. Si las dos sumas coinciden el archivo ha sido transmitido sin errores, sin embargo si no coincide es posible que haya sido corrompido.

El último campo obligatorio de la cabecera es el puntero a urgentes. En sus 16 bits de longitud da la posición de los datos considerados como urgentes. Como se ha explicado antes, este campo solo se lee en caso de que el Flag URG esté activo, en caso contrario se ignorarán los datos contenidos en él, por lo que podrán ponerse a cero. Este campo será el último de la cabecera a no ser que esta cuente con el campo Opciones. Este apartado, cuando exista, tendrá una longitud de 32 bits o múltiplos de esta cifra. De esta manera, se consigue que la cabecera pueda ser separada de los datos en palabras de 32 bits, ya que de otra manera sería más problemático. La longitud de este campo viene data por el campo en el que se especificaba la longitud de la cabecera. Si los datos

contenidos en este campo no llegaran a la longitud necesaria para completar la palabra de 32 bits, se rellenarán los bits sobrantes con ceros. La información que contiene este apartado puede ser de diversa índole, siendo habitual que haga referencia al escalamiento de la ventana de recepción, a la interpretación del checksum o a la longitud de paquete máxima que puede ser transmitida por la red (MTU).

2.3.3 El segmento TCP: Datos

Una vez finalizada la cabecera se pueden encontrar los datos, también conocidos como payload. El formato de la parte de datos dependerá solamente de la aplicación de la que provengan, ya que el protocolo TCP no los modifica. Antes de comenzar a reconocer la transferencia de los datos propiamente dicha se recibió información de que, en el caso del E.reader, los datos deberían seguir el protocolo Universal Data Bin File Format, sin embargo como se verá a continuación esto no es así. De todas formas, a fin de conocer el tipo de paquete que se buscó en un principio, es interesante tener unas nociones básicas de este formato de datos. No existe mucha información sobre este formato de datos más allá de un archivo de la propia Gantner Instruments, lo que hace pensar que quizá sea un protocolo desarrollado por ellos mismos para sus productos. En el citado documento, se afirma que el formato puede usarse para almacenar datos en forma binaria de una manera eficiente. En ese mismo archivo diferencian cuatro partes en la estructura del formato: Cabecera, caracteres de separación, sección de datos y checksum.

2.3.3.1 Cabecera UDBF

Debido a la gran cantidad de parámetros que están contenidos en la parte de la cabecera, solo se explicarán los más relevantes. Dado que puede resultar interesante conocer el resto de datos, a continuación se puede observar la lista completa de los bits de la cabecera y sus cometidos tal como a parecen en el documento original del Universal Data Bin File Format:

7.2.1. Header

IsBigEndian (1 byte unsigned integer)
0 ..... IsLittleEndian (like Intel CPU's)
<>0 ..... IsBigEndian (like Motorola CPU's)
Attention: If the decoding CPU has a different "Endian" as marked in this file, it needs to swap all data bytes, to get right access. Only 1 byte data and for this also 1 byte data array like strings don't need to be swapped.
Swapping must be done like: e.g. 4 byte integer which is B0 B1 B2 B3 => B3 B2 B1 B0

Version (2 byte unsigned integer)
Version of file structure times 100. E.g. dec. 110 means version 1.10.
Actual version: 1.07

TypeVendorLen (2byte unsigned integer)
Changed from V1.02 to V1.06
Delivers number of 1byte integer's, which are used for TypeVendor.

TypeVendor (1byte unsigned integer array)
Changed from V1.02 to V1.06
TypeVendor including terminating \0.
This is now: "UniversalDataBinFile - GANTNER instruments"

WithCheckSum (1 byte unsigned integer)
Changed from V1.00 to V1.01
0 ..... NoCheckSum
<>0 ..... WithCheckSum

ModuleAdditionalDataLen (2 byte unsigned integer)
Shows how many ModuleAdditionalData bytes are appended. If a ModuleAdditionalDataLen is set, it needs to be at least 4 bytes in length in order to declare variables MODULETYPE and MODULEADDITIONALDATASTRUCTID to determine how ModuleAdditionalData is coded.

Optional: ModuleType (2 byte unsigned integer)
If ModuleAdditionalData is appended, this item needs to be set first and is the type of module in standard coding.

Optional: ModuleAdditionalDataStructID (2 byte unsigned integer)
If ModuleAdditionalData is appended, this item needs to be set second and is an ID to describe used structure.

Optional: ModuleAdditionalData
If ModuleAdditionalDataLen is set to >4, all following data depend on a user-definable structure. How this structure is coded depends on the version number and is described with MODULETYPE and MODULEADDITIONALDATASTRUCTID.

StartTimeToDayFactor (double – 8 byte IEEE754 number)
This is the factor of the following StartTime with respect to a day.
StartTime*StartTimeToDayFactor = Time in [days]

dActTimeDataType (2 byte unsigned integer)
Added in V1.07
Delivers data type of ActTime in standard coding. Refer to "DataType" parameter of variable below.
If this is set to "No" than no ActTime is included.

dActTimeToSecondFactor (double – 8 byte IEEE754 number)
This is the factor of the following dActTime (in time/data section) with respect to a second.
dActTime*dActTimeToSecondFactor = Time in [s]
If this is set to a value <=0.0 than no ActTime is included.

StartTime (double – 8 byte IEEE754 number)
This is the time at which the startevent occurred. This time is base for all following data items (dActTime)

SampleRate (double – 8 byte IEEE754 number)
This is the measurement frequency.

VariableCount (2 byte unsigned integer)
Number of configured variables.

Variable settings:
Will be looped for as long as indicated by VariableCount.

NameLen (2 byte unsigned integer)
Delivers number of 1byte integer's, which are used for VariableName.

Name (1 byte unsigned integer array)
Name of variable including terminating \0.

DataDirection (2 byte unsigned integer)
Delivers data direction of variable in standard coding.

Input..... 0
Output..... 1
InputOutput..... 2
Empty..... 3

Attention: If a variables data direction has no "Input" set, it also isn't be set in time/data section!

DataType (2 byte unsigned integer)
Delivers data type of variable in standard coding.

Changed from V1.01 to V1.02:
No 0
Boolean.....1
SignedInt8.....2
UnSignedInt8.....3
SignedInt16.....4
UnSignedInt16.....5
SignedInt32.....6
UnSignedInt32.....7
Float 8
BitSet8 9
BitSet16 10
BitSet32 11
Double .....12 Added in V1.07

FieldLen (2 byte unsigned integer)
Delivers the set field length.

Precision (2 byte unsigned integer)
Delivers the set precision (see also DataType).

UnitLen (2byte unsigned integer)
Changed from V1.02 to V1.06
Delivers number of 1byte integer's, which are used for Unit.

Unit (1byte unsigned integer array)
Changed from V1.02 to V1.06
Unit including terminating \0.

AdditionalDataLen (2 byte unsigned integer)
Tells how many AdditionalData bytes are appended. If an AdditionalDataLen is set, it needs to be at least 4 bytes to declare variables TYPE and ADDITIONALDATASTRUCTID to determine how AdditionalData is coded.

Optional: AdditionalDataType (2 byte unsigned integer)
If AdditionalData is appended, this item needs to be set first and shows the type of variable in standard coding.

Optional: AdditionalDataStructID (2 byte unsigned integer)
If AdditionalData is appended, this item needs to be set second and is an ID to describe used structure.

Optional: AdditionalData

If AdditionalDataLen is set to >4, all following data depend on a user-definable structure. How this structure is coded depends on the version number and is described with TYPE and ADDITIONALDATASTRUCTID.

AdditionalDataType = 11 (SlaveVariableType):
AdditionalDataStructID = 1:
AdditionalDataLen 10
AdditionalData Item 1:.....UARTIndex (2 byte unsigned integer)
AdditionalData Item 2:.....SlaveAddress (2 byte unsigned integer)
AdditionalData Item 3:.....SlaveDataIndex (2 byte unsigned integer)

Los nueve primeros bytes corresponden a información del sistema, como el fabricante y la versión del software. Entre ellos, también está contenida información a cerca de la forma en que se almacenarán los datos en el disco (Endianness). Tras esto, se puede encontrar una serie de datos opcionales, que tendrán longitud variable y que informarán sobre el módulo.

Los apartados desde StartTimeToDayFactor hasta SampleRate son los que informan sobre el comienzo de la captura, la duración de la misma y la cantidad de capturas por segundo. Esta información es muy importante, ya que se utilizará para diferenciar cada dato capturado.

Los siguientes parámetros de la cabecera hacen referencia a las variables, y salvo VariableCount, que es el que informa sobre el número de variables configuradas, son opcionales. El número de parámetros vendrá determinado por VariableCount. Entre estos parámetros se encuentran el nombre de la entrada (junto con la longitud reservada para el mismo), la dirección de los datos (si es una entrada, una salida o ambas) y el tipo de datos asignado a dicha entrada. En cuanto a esto último (el tipo de datos), existe un listado con las equivalencias entre el valor de dos bits sin signo que consta en DataType y su traducción. Según este listado el número 1 corresponde a datos de tipo Booleano

(Verdadero o falso), el número 4 a palabras de 16 bits de números enteros y con signo y el número 8 a datos del tipo coma flotante. En este caso concreto, solo se pueden mostrar 12 tipos de datos, que son los que se muestran en la tabla de equivalencias. Este número puede ser incrementado en versiones siguientes, ya que como consta en el documento, en la versión anterior solo se reconocían 6 tipos de datos: Booleano, Entero (Sint16), Real (Coma flotante), Bitset8, Bitset16 y Entero largo (Sint32). Otros parámetros de este mismo tipo son la precisión (número de dígitos después de la coma) o las unidades de los datos, para la que se ha asignado un byte. Además de esto existen más datos opcionales, que se pueden activar y configurar según las necesidades.

Si se realiza la suma de los espacios reservados para todos estos parámetros de la cabecera, se obtiene que la longitud mínima de la misma es 35 bytes, lo que corresponde a 280 bits o 70 caracteres hexadecimales. Esto es importante, ya que a la hora de buscar paquetes de este tipo en la comunicación, este será el modelo a seguir.

2.3.3.2 Caracteres de separación UDBF

Este es otro punto que ayudará sobre manera a la identificación de este tipo de paquetes. La separación entre la cabecera y los datos, se realiza en este caso mediante unos caracteres que tendrán forma de asterisco '*' y que no serán en ningún caso menor número que ocho. No es posible determinar la longitud exacta de la separación ya que vendrá determinada por los campos opcionales de la cabecera. Sin embargo, según el documento los datos deberán comenzar siempre una nueva palabra de 16 bytes. Esta estructura, al ser algo inusual, debería ser fácilmente reconocible.

2.3.3.3 Datos UDBF

En esta parte simplemente se enumeran los datos. En el documento no se da información sobre la separación entre ellos, por lo que esto no podrá servir de ayuda para el reconocimiento. Si que se especifica que cada dato irá acompañado de su signatura de tiempo (que tiene por nombre dActTime). El dato dActTime, que no se encuentra en la cabecera, se puede derivar de dActTimeToSecondFactor ya que este sí se conoce. Este dato, es del tipo double y su representación se llevará a cabo siguiendo el estándar IEEE 754, que es que se utiliza para representación de números en coma flotante. Se buscarán este tipo de datos dentro de los paquetes.

2.3.3.3.1 Interpretación de datos IEEE 754

El estándar IEEE 754, especifica como se deben representar números reales en palabras de 32 bits. También existen extensiones del estándar para representación en palabras de 64 y 128 bits. Los números reales que es posible representar mediante este estándar son tanto positivos como negativos y están comprendidos entre el 1×10^{-101} y el 9999999×10^{90}. La representación consta de varios pasos, a fin de que la explicación resulte más sencilla se supondrá que se dispone del número ya representado en su notación en bits y se desea obtener su valor decimal. En la conversión se trata de en primer lugar convertir la notación en bits a formato de coma flotante normalizado (como por ejemplo $1{,}110110101\cdot2^{6}$) y posteriormente realizar la conversión a decimal.

Por lo tanto, los primeros pasos estará encaminados a interpretar el signo, la mantisa y el exponente de la notación de coma flotante.

En primer lugar, se parte de la representación binaria del número, por ejemplo:

0100 0100 1101 0010 1010 1001 0000 0000.

El primer bit de la cadena corresponde al signo. Si es un cero el número será positivo, sino será negativo. En este caso es cero, lo que significa que se trata de una cifra positiva.

Los siguientes 8 bits (bits 2 a 9) corresponden al exponente, que en este caso serían:

1000 1001

Si se traducen estos bits a decimal se obtiene el número 137. Para obtener el exponente se debe restar 127 al número obtenido, quedando el exponente en 10. Con esto ya se conoce la representación normalizada:

$$1.10100101010100100000000 \cdot 2^{10}$$

Si desplazamos diez posiciones desde el bit en que terminaba el exponente se podrá comenzar a traducir el número:

11010010101.0100100000000

En este punto se comienza la conversión de la manera habitual entre binario y decimal, es decir multiplicando el valor de cada bit por su potencia de dos correspondiente, obteniendo el número 1685.28125. La operación sería la siguiente:

$$1 \cdot 2^{10} + 1 \cdot 2^{9} + 1 \cdot 2^{7} + 1 \cdot 2^{4} + 1 \cdot 2^{2} + 1 \cdot 2^{0} + 1 \cdot 2^{-2} + 1 \cdot 2^{-5} = 1685.28125$$

De esta manera se consigue la conversión entre la representación en coma flotante a decimal. Si se requiriera la conversión inversa, que en este caso no será necesaria ya que se trata de comprender valores escritos en coma flotante, se deberán seguir los mismos pasos en el orden contrario al aquí expuesto.

2.3.3.4 Checksum UDBF

Esta parte, la final del paquete, es opcional. Su existencia viene dada por el byte WithCheckSum, que se puede encontrar entre los primeros de la cabecera. Si dicho byte es igual a cero (0000 0000) esta parte no aparecerá, sin embargo si el byte es distinto de cero (supuestamente cualquier valor) el checksum estará activo. Esto es importante ya que de otra manera podría confundirse esta suma con el final de los datos. En caso de que el checksum esté activado, este se calculará a partir de cada uno de los bytes anteriores a la propia suma y tendrá una longitud fija de 4 bytes en un formato de número entero sin signo.

2.3.4 Transmisión de datos real

2.3.4.1 Análisis: Primer segmento

Como se ha explicado anteriormente, la transferencia TCP comienza con un Three Way Handshake. Tras esto comienzan unas repeticiones de envíos que siguen la siguiente estructura: Se envía un paquete con los flags PSH y ACK del host (software) al servidor (módulo). El servidor responde enviando un datagrama con los mismos flags activados. Una vez hecho esto, el host vuelve a enviar otro paquete similar al primero, al que se responde con un paquete con una longitud mucho mayor al anterior. Una vez recibido este último datagrama, el host envía un ACK. Esta misma estructura de envíos se repite cada medio segundo aproximadamente. Por lo tanto, se tienen tres paquetes enviados por el host y dos por el servidor. El primero tiene esta estructura:

```
0000  00 12 08 09 c4 8d 00 b0  d0 e1 59 cb 08 00 45 00   ........ ..Y...E.
0010  00 33 88 7c 40 00 80 06  ec d6 c0 a8 02 0f c0 a8   .3.|@... ........
0020  02 12 07 4c 1f 41 bf a8  bf 89 21 bf f6 ba 50 18   ...L.A.. ..!...P.
0030  ff ff 85 97 00 00 00 09  00 00 00 00 00 00 00 00   ........ ........
0040  4b                                                K
```

Para la correcta interpretación del datagrama es importante resaltar que cada par de valores hexadecimales corresponde a un byte. Los primeros 14 bytes corresponden a la cabecera ethernet adjuntada al paquete en la capa de enlace del modelo OSI. De estos 14 bytes, los primeros 6 (00 12 08 09 c4 8d) corresponden a la dirección MAC del equipo destino (en este caso el E.reader) y los siguientes 6 (00 b0 d0 e1 59 cb) son la representación de la dirección MAC del equipo de origen. Los últimos dos bytes de esta cabecera (08 00) dan el protocolo de la capa de red, que en este caso es IP.

Los siguientes 20 bytes corresponden a la cabecera IP, añadida en la capa de red del modelo OSI. El primer byte (45), otorga dos informaciones distintas, por un lado da la versión del protocolo (en este caso versión 4) y por otro da la longitud de la cabecera en palabras de 32 bits (5 palabras o 20bytes). El siguiente byte se utiliza para marcar preferencias en el envío del paquete, dado que su valor es nulo, no hay ninguna. Los siguientes dos bytes (00 33), que traducidos a decimal tienen un valor de 51, definen la longitud total del datagrama IP en bytes (lo que no incluye la cabecera ethernet). Los bytes 5 y 6 son un identificador de paquete. Los dos bytes que los siguen (40 00) corresponden a los flags (40) y al offset (00) del paquete. Este valor de los flags informa da que el datagrama no debe ser fragmentado y por ello el offset siempre será cero. El siguiente byte (80) confirma que el paquete no ha pasado por ningún router y el sucesivo (06) define el protocolo que usará la capa de transporte y que como se ha visto será TCP. Los bytes *ec* y *d6* corresponden al checksum de la cabecera. Los último ocho bytes son las direcciones IP de origen y de destino respectivamente y tienen una longitud de 4 bytes cada una.

La última cabecera anterior a los datos es la correspondiente al protocolo TCP y tiene una longitud de 20 bytes. Los primeros dos bytes (07 4c) son el número de puerto de origen y los dos siguientes (1f 41) los que corresponden al destino. Los ocho bytes

siguientes son el número de secuencia y de confirmación respectivamente. Tras ellos se encuentran la longitud de la cabecera (50) y los flags (18), que corresponden a un PSH y un ACK activos. Por último, se encuentran la longitud de la ventana y el checksum.

Los restantes 11 bytes corresponden a los datos, pero no parece que haya mucha información en ellos. Probablemente se trata del código para la petición de datos, ya que el módulo responde de esta manera. No parece que cumpla las normas del estándar UDBF, ya que su longitud es mucho menor que la que se podría esperar de un paquete regido por dicho estándar. Su trascripción en ASCII tampoco es de ayuda. Sin embargo, la información de este paquete es igual a la que se puede encontrar en los primeros paquetes de cada repetición o bucle, lo que parece indicar que esta información es la causante del inicio de cada envío de datos.

2.3.4.2 Análisis: El segmento de datos

2.3.4.2.1 Cabecera TCP

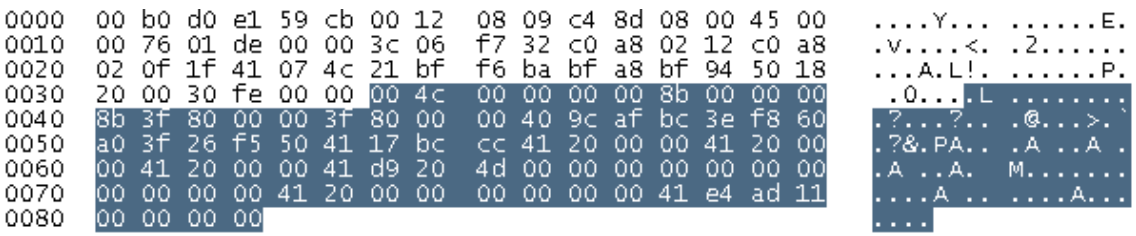
El paquete de respuesta tiene la estructura que sigue:

```
0000  00 b0 d0 e1 59 cb 00 12  08 09 c4 8d 08 00 45 00   ....Y... ......E.
0010  00 76 01 de 00 00 3c 06  f7 32 c0 a8 02 12 c0 a8   .v....<. .2......
0020  02 0f 1f 41 07 4c 21 bf  f6 ba bf a8 bf 94 50 18   ...A.L!. ......P.
0030  20 00 30 fe 00 00 00 4c  00 00 00 00 8b 00 00 00    .0....L ........
0040  8b 3f 80 00 00 3f 80 00  00 40 9c af bc 3e f8 60   .?...?.. .@...>.`
0050  a0 3f 26 f5 50 41 17 bc  cc 41 20 00 00 41 20 00   .?&.PA.. .A ..A .
0060  00 41 20 00 00 41 d9 20  4d 00 00 00 00 00 00 00   .A ..A.  M.......
0070  00 00 00 00 41 20 00 00  00 00 00 00 41 e4 ad 11   ....A .. ....A...
0080  00 00 00 00                                       ....
```

Como en el caso anterior, los primeros 14 bytes corresponden a la cabecera de ethernet. Las direcciones son las mismas que en el paquete anterior pero el origen y el destino se han intercambiado. Obviamente el protocolo de la capa de red es el mismo.

La cabecera IP también tiene pocas variaciones con respecto al paquete anterior. La versión del protocolo, la longitud de la cabecera y las preferencias son iguales. Hasta aquí llegan las similitudes, ya que el resto de parámetros cambian. La longitud total del datagrama es ahora 118 bytes, el número de identificador es distinto y no hay ningún flag activado. El TTL informa de que el paquete no ha pasado por ningún router, pero su valor es menor que en el paquete anterior. Por último, las direcciones IP, como ocurría con las direcciones MAC, han cambiado de posición, siendo ahora el destino el software y el origen el módulo.

La cabecera TCP sigue las mismas directrices que la anterior, cambiando tan solo los números de secuencia y de confirmación y el puerto de origen y de destino. Los 78 bytes restantes corresponden a los datos:



A diferencia del paquete anterior en este si que parece haber cierta información, aunque su longitud parece algo escasa para cumplir el estándar UDBF. Para ver si efectivamente se cumple el estándar, se habrá de hacer coincidir byte a byte la información del paquete con la especificada en el documento del UDBF.

2.3.4.2.2 Comparación con estándar UDBF

Para comprobar si es posible que haya una cabecera incluida en estos datos, se sabe que la longitud mínima de la cabecera de UDBF son 35 bytes, dado que la cantidad de bytes de datos es 78 es posible que esté contenida en ellos. El primer byte indicaría que los datos se almacenarán en Little Endian (que es el formato típico para CPUs Intel). Los siguientes dos bytes, harían referencia a la versión. Según el documento anteriormente estudiado, la trascripción en bits del valor aquí contenido dividido entre cien daría como resultado la versión actual. Como ejemplo de esto, se da el valor 110 y se especifica que se referiría a la versión 1.10. Dado que hay dos bytes dedicados a este fin (4c 00), la trascripción quedaría de la siguiente manera: 0100 1100 0000 0000. No es posible convertir este número de la manera que se explica en el documento y que el resultado tenga sentido. Incluso si se desecharan los ceros anteriores y posteriores, se tendría lo siguiente 10011, lo que daría que la versión actual es la 100.11. Obviamente, existen problemas de concordancia entre el protocolo y el paquete.

Si se sigue analizando el paquete, el siguiente par de bytes está destinado a especificar la longitud del nombre del fabricante. Su valor es cero, lo que indica que no hay espacio reservado a tal efecto, sin embargo este espacio no es opcional y por lo tanto tiene que aparecer con una longitud mínima de 1 byte.

El siguiente byte especifica si el checksum está activo o no, el valor no es cero, por lo que sí está activo. Los dos bytes que se encuentran tras el indicador de checksum son los utilizados para dar la longitud de la primera parte de datos adicionales. Al ser cero, se entiende que no se incluirá esta información añadida.

Los siguientes ocho bytes, conforman un número acorde con IEEE 754 que actuará como clave de tiempos. Este debería corresponder a un número en coma flotante de 64 bits. Su representación hexadecimal es la siguiente: 00 8b 3f 80 00 00 3f 80. La traducción de este número se hace de manera similar a la explicada anteriormente para representaciones de 32 bits. Para hacer la conversión se ha de tener en cuenta que los bits que corresponden al cálculo del exponente serán 11 y para la mantisa pasarán a ser 52 y el nuevo desplazamiento no será 127 sino 1023. Teniendo esto en cuenta, no es posible hallar la trascripción del número ya que el valor del exponente sería -1007 y no se dispone de tal número de dígitos para el desplazamiento. Esta es una nueva

discordancia entre el estándar y el paquete que hace razonable pensar que no se está ante información empaquetada de la manera esperada.

Dado que la cabecera no coincide con el estándar, se buscarán otras posibles coincidencias dentro del paquete, como por ejemplo los caracteres de separación. En el documento, se especifica que han de ser una serie de asteriscos con una longitud no menor de 8 y que la longitud de los datos que los precedan debe ser múltiplo de 16 bytes. Si se buscan los asteriscos en el paquete no se obtendrán resultados, sin embargo existe una cadena de 11 bytes de ceros a la que le sigue una cadena de datos de 16 bytes exactamente. Es posible que esta cadena de ceros sean los caracteres de separación, para comprobarlo, se probará si los datos que los siguen tienen formato de coma flotante, ya que deberían corresponder al dActTime, que tiene este formato. El primer dato que se puede encontrar es 41 20 00 00 00 00 00 00. Si se realiza la conversión el resultado es 524288, que aunque no aparenta ser un dato de tiempo, al menos es posible realizar el cambio a decimal.

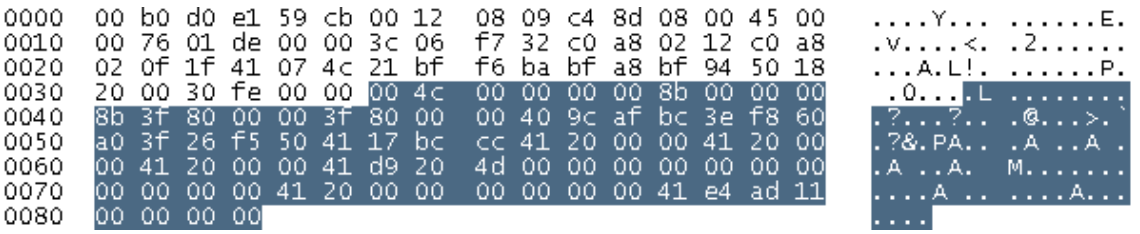
2.3.4.2.3 Conversión real de valores

A modo de prueba, también se puede realizar la conversión tomando tan solo los primeros 32 bits (4 bytes) del número. El resultado de esta conversión es algo sorprendente, ya que en binario es 1010.0 o lo que es lo mismo 10 en decimal. Este valor es el que aparece por defecto en las entradas configuradas para recibir voltaje que no tienen sensores conectados a ellas. Si se analiza de esta misma manera el siguiente dato (41 e1 ad 11) el resultado es 28.207, que coincide exactamente con el valor de uno de los sensores de temperatura conectados al sistema. Si se siguen analizando el resto de valores de la parte de datos, se puede comprobar que todos ellos coinciden con los valores que se están captando en el momento, como por ejemplo 41 d9 20 4d cuya traducción a decimal es 27.14 y que corresponde a un segundo sensor de temperatura o el valor *40 9c af bc* cuya trascripción decimal es 4.89 y que corresponde a la medición de presión de uno de los sensores.

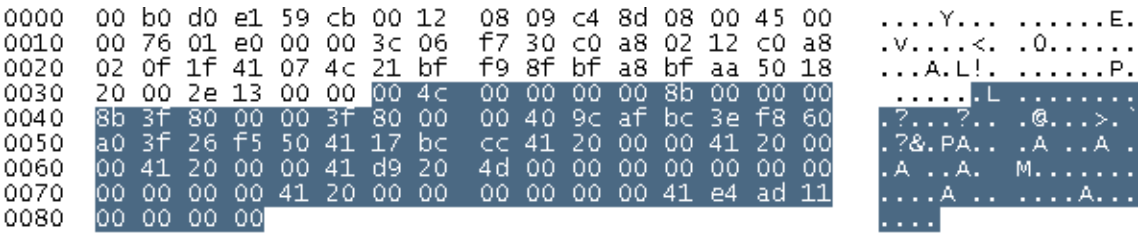
2.3.4.2.4 Otra información

Por lo tanto, se sabe que los valores capturados se transmiten en este paquete, y que no cumple el estándar que se pensaba. Sin embargo, el inicio de los datos no está del todo claro, ya que aparecen varios dígitos que no han sido identificados aún. Si se compara este paquete con los de las siguientes repeticiones, se pueden sacar algunas conclusiones interesantes. Aquí se pueden comparar tres paquetes de datos de tres repeticiones consecutivas:

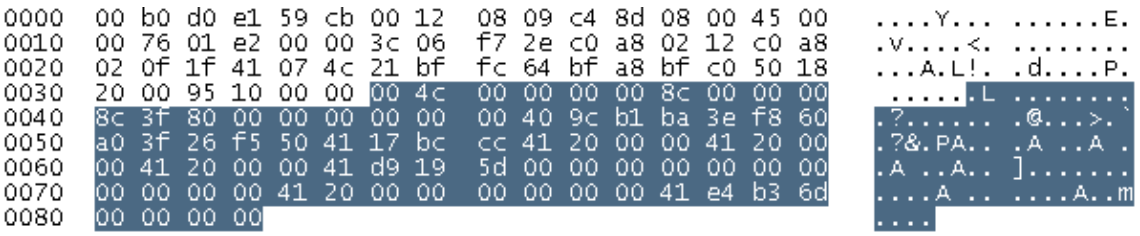
Primer paquete:



Segundo paquete:



Tercer paquete:

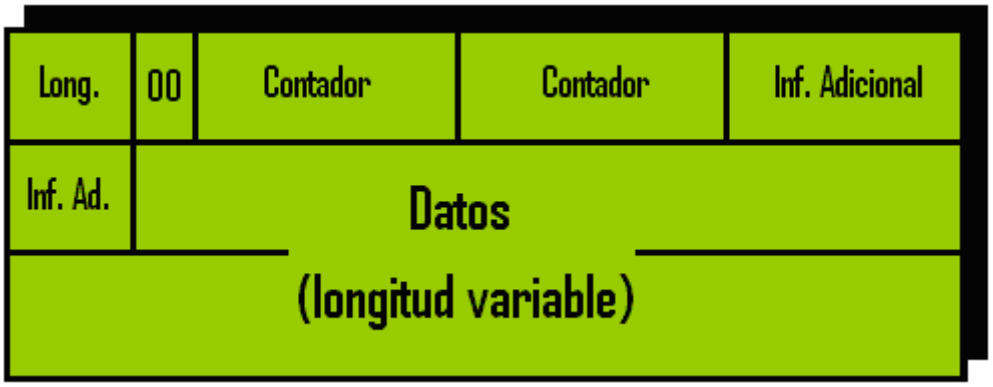


Como puede comprobarse, el primer y segundo paquetes son iguales. Por lo tanto, aunque se envíen datos cada medio segundo, estos se repiten, dejando el intervalo de captura final en un segundo. Los tres paquetes tienen su parte de datos de igual longitud y comenzando por 00 4c. Si se convierte a decimal se pude comprobar que estos dos bytes dan la longitud de los datos, que en este caso es 76. Existe un desfase de dos bytes entre la longitud real y la que aquí consta, que es debido a que no se tiene en cuenta el espacio de los propios bytes usados para indicar la longitud.

Después de estos bytes se encuentran un serie de ceros, seguidos por un byte con datos, que a su vez es precedido por 3 bytes de ceros y un nuevo byte con datos igual al anterior. En el primer y segundo paquetes estos bytes de datos son iguales (8b) y su trascripción en decimal es 139. Sin embargo, en el tercer paquete esto cambia, siendo el nuevo valor 8c ó lo que es lo mismo 140. Por lo tanto, el valor aumenta en una unidad cada nueva captura. Si se observa este mismo byte en sucesivos paquetes, se confirma que pertenece a un contador o identificador de captura que va incrementándose uno a uno. Es importante resaltar que no se trata de un identificador de paquete ya que, como se ha visto, si la información de la captura no cambia el valor tampoco lo hace. Probablemente, el valor real del contador sea *00 00 00 8b*, siendo por lo tanto 4.294.967.295 (ff ff ff ff) el máximo valor posible de capturas, lo que equivaldría a

49710 días o 136 años aproximadamente, con la frecuencia de muestreo actual. Este es el tiempo que podrá funcionar el E.reader sin necesidad de ser reiniciado.

Tras esto, llegan dos bytes con datos seguidos por dos bytes de ceros. Los cuatro bytes de los que se habla son los siguientes *3f 80 00 00*. A continuación estos bytes se repiten y comienzan los datos propiamente dichos. Estos bits son iguales para los siguientes paquetes, lo que puede indicar que se trate de información sobre el formato de los datos u otro tipo de información fija. Por lo tanto el paquete de datos quedaría dividido de la siguiente manera:



La cabecera tiene una longitud de 19 bytes. Los dos primeros están reservados para la longitud del paquete. El siguiente byte actúa de separador y no contiene información. Tras este, se encuentran los dos contadores, que son idénticos y que tienen una longitud de 4 bytes cada uno. Por último, se encuentra la información no descifrada a la que se ha nombrado como adicional. El espacio dedicado a esta es de 8 bytes. Siguiendo a la cabecera se encuentran los datos propiamente dichos, que tienen una longitud variable. Los valores se encuentran en formato de coma flotante de 32 bits (4 bytes) sin separaciones entre ellos.

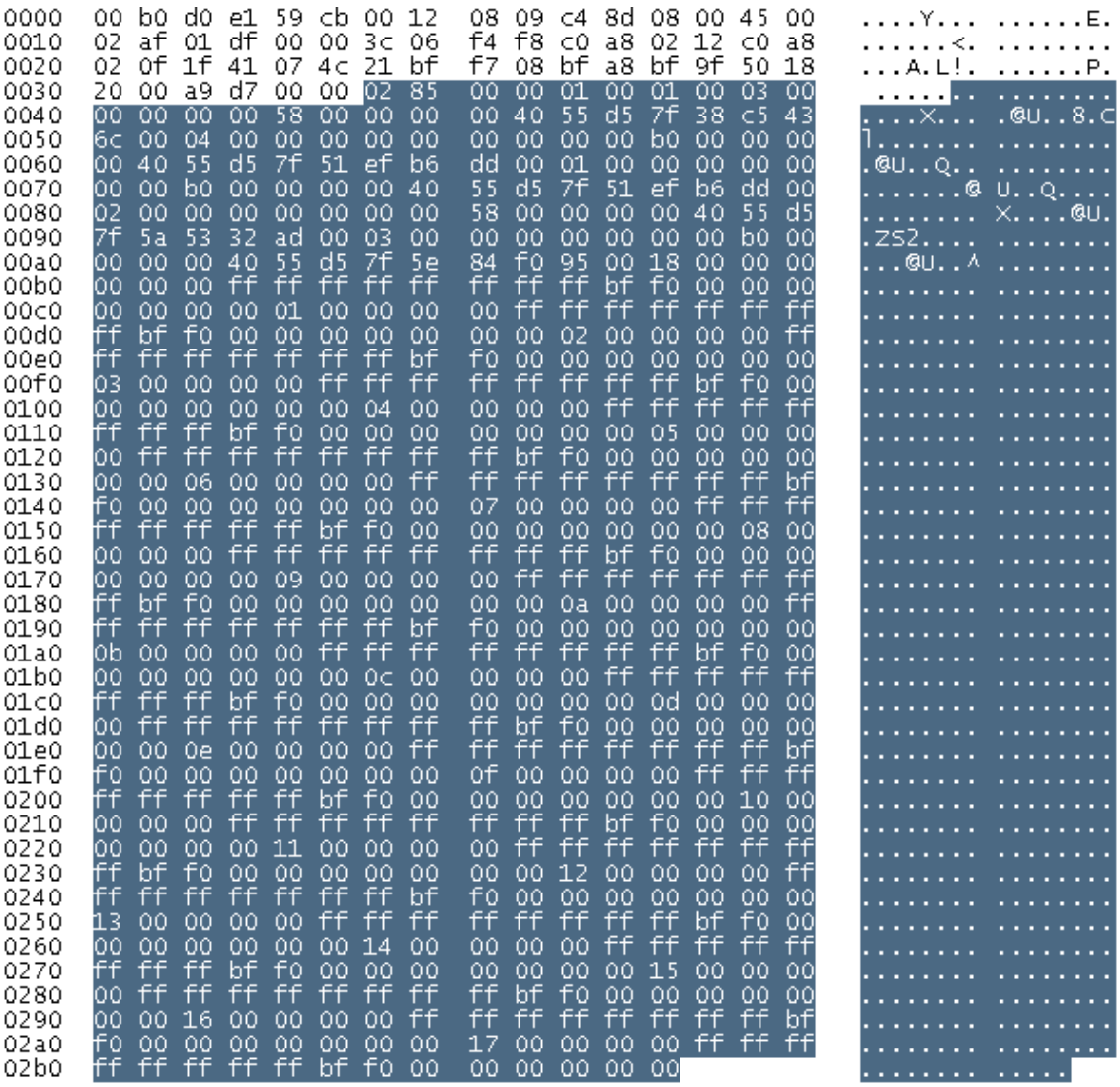
2.3.4.3 Análisis: Segmentos siguientes

Una vez recibido el paquete de datos el Host envía un nuevo paquete muy similar al inicial:

```
0000  00 12 08 09 c4 8d 00 b0  d0 e1 59 cb 08 00 45 00   ........ ..Y...E.
0010  00 33 88 7d 40 00 80 06  ec d5 c0 a8 02 0f c0 a8   .3.}@... ........
0020  02 12 07 4c 1f 41 bf a8  bf 94 21 bf f7 08 50 18   ...L.A.. ..!...P.
0030  ff b1 85 97 00 00 00 09  0b 00 00 00 00 00 00 ff   ........ ........
0040  ff                                                .
```

Las únicas diferencias entre los dos datagramas enviados por el software, a parte de la diferencia obvia de numeración de paquete, se encuentran en el payload del datagrama. En el paquete inicial los datos son los siguientes 00 09 00 00 … 00 00 4b, sin embargo en este los datos cambian a 00 09 0b 00 … 00 ff ff. En ambos paquetes el payload comienza con 00 09, que se refiere a la longitud en bytes de los datos presentados a continuación. La función del resto de los bytes no es del todo clara, aunque sí se puede afirmar que utilizando los bytes del primer paquete se obtienen los datos. El contenido

del paquete de respuesta al segundo datagrama tampoco está definido. Aquí puede verse el paquete de respuesta completo con el payload resaltado:



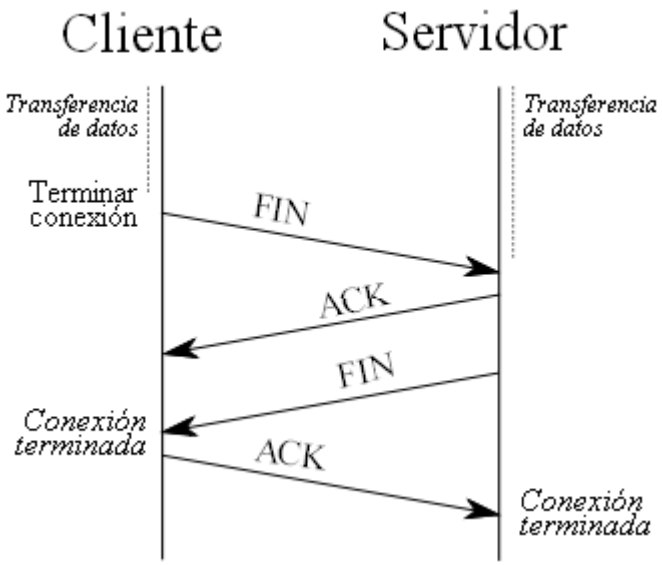
Como puede comprobarse, su longitud es mucho mayor a la de los paquetes anteriores. Como es habitual, los dos primeros bytes del payload indican la longitud de los datos sucesivos, que en este caso es de 645 bytes. Los siguientes 12 bytes no cambian en ninguno de los envíos. El byte que los sigue sin embargo, tiene un comportamiento peculiar, ya que en el primer paquete de este tipo (el aquí mostrado) tiene un valor de 58 pero en los sucesivos envíos alterna entre el valor 00 y el valor 01. Le siguen cuatro bytes con valor cero. Hasta aquí, los datos podrían considerarse como de cabecera.

Los bytes que siguen a esta parte pueden dividirse en dos partes según la estructura. La primera parte consta de datos no numerados agrupados en palabra de 8 bytes probablemente con formato de coma flotante. La segunda parte es una serie numerada de datos del mismo valor y 10 bytes de longitud cada uno separados por ceros. Entre los datos no numerados de la primera parte se pueden encontrar unos dígitos que parecen ser de control, como lo son el tercer byte de la línea 0050, el decimoprimer byte de la

línea 0060, el primer byte de la línea 0080 y el séptimo byte de la línea 0090. Los datos de esta primera parte tienen todos valores en decimal muy similares, en este caso rondando los 87.3. Los siguientes paquetes siguen esta misma tendencia pero con distintos valores, aunque todos ellos cercanos a 1. La última parte consta de 24 repeticiones de la misma cadena de bytes. La función de este paquete no está clara y sus datos parecen redundantes. Teniendo en cuenta que los valores capturados por los sensores se pueden encontrar en un paquete anterior, la comprensión de la información contenida en este datagrama no será necesaria, aunque convendrá conocer su estructura básica en caso de que se encuentren problemas relacionados con él en un futuro. El último paquete del bucle es simplemente un datagrama con el Flag ACK activo que servirá para indicar que se ha recibido toda la información. Este paquete no contiene datos.

2.3.4.4 Desconexión TCP

Este bucle de cinco paquetes se repite cada medio segundo durante el tiempo que dure la captura. La finalización de la comunicación la pondrá en marcha el cliente cuando ya no sea necesario seguir capturando valores. El protocolo TCP tiene prevista una rutina para la desconexión, que se conoce como Four Way Handshake. El esquema típico es el que se puede ver en el grafico siguiente:



El cliente comienza enviando un paquete con el Flag FIN activado y espera a la respuesta. Cuando el servidor recibe el paquete de FIN envía a su vez dos nuevos datagramas, el primero para confirmar la recepción del paquete y el segundo para aceptar el cierre de la comunicación. Cuando el cliente recibe este segundo paquete cierra la conexión, no sin antes enviar un último datagrama de confirmación, al que el servidor esperará para cerrar la conexión. Si este último paquete se extraviara y no llegara al servidor, este quedaría inutilizable ya que no aceptará una nueva conexión sin haber terminado la antigua. Para minimizar esto, el servidor deberá tener configurado un time out de unos segundos, tras los cuales la conexión será cerrada unilateralmente por él mismo.

Una vez se han estudiado los rudimentos de la comunicación que se habrá de replicar, será necesario conocer las herramientas que existen para implementar una interfaz de comunicación que satisfaga las necesidades impuestas

3 Introducción a C#

3.1 Conceptos básicos

El lenguaje de programación C# y su entorno de desarrollo Visual Studio, serán los elegidos para implementar la interfaz. Este lenguaje de reciente creación forma parte de una nueva corriente llamada programación orientada a objetos (POO), que se adecua muy bien a las necesidades de desarrollo de interfaces gráficas. Además forma parte de la plataforma .NET impulsada por Microsoft, que pretende mejorar la interoperabilidad de software de distintas empresas.

La programación orientada a objetos está basada en una serie de conceptos nuevos para una persona no iniciada en la programación. En la programación clásica las aplicaciones tienen una estructura lineal fuertemente arraigada al procesamiento de datos, donde se inserta un dato y al finalizar se obtiene un resultado. Con la POO el programa tendrá una estructura modular en la que se darán eventos que harán desarrollarse el programa de maneras distintas. Para poder explicar las estructuras básicas de este tipo de programación es necesario presentar unos conceptos básicos, que también ayudarán a comprender el desarrollo de la interfaz, como son los siguientes:

Objeto: Son las variables del programa. Tienen propiedades y atributos asociados y se utilizan para llevar a cabo rutinas. A diferencia de las subrutinas clásicas, los objetos pueden interactuar entre ellos y responder a eventos.

Clase: Son el ente que contiene las definiciones de los objetos. Todo objeto pertenece a una clase. Los objetos basados en una misma clase también compartirán las propiedades y atributos que provengan de dicha clase.

Herencia: Es una parte fundamental de la POO y la que permite modificar un programa sin reescribirlo. Los objetos heredan las características de la clase y por lo tanto toda modificación de las propiedades de la misma se verá reflejada en los objetos derivados de ella.

Método: Es un algoritmo asociado a un objeto o una clase. Para que comience su ejecución se ha de accionar un evento o recibir un “mensaje”. El hecho de accionar un evento podría formar parte de otro método.

Evento: Es similar a lo que en informática se conoce como una interrupción. Habitualmente es generada debido a la interacción de un usuario con el programa. Es la principal herramienta para la ejecución de métodos.

Mensaje: Es una comunicación dirigida a un objeto. También puede usarse para desencadenar la ejecución de un método.

Propiedad: Son las características de un objeto o clase. Diferencian a los objetos entre sí. El valor de las mismas puede ser alterado utilizando un método.

Existen numerosos textos para aproximarse al estudio de la POO, por lo que no se explicarán en profundidad los conceptos teóricos en los que se basa. De todas formas, si que podría resultar interesante conocer como se estructuran los programas diseñados con este tipo de programación en mente. Para ello, después de estudiar algunos de los textos citados anteriormente, se inició una serie de tutoriales a fin de adquirir unos conocimientos prácticos. A continuación se expondrán algunos de los ejercicios básicos de estos turoriales que servirán como iniciación a la POO y ayudarán a comprender mejor las explicaciones sobre el diseño de la interfaz final.

3.2 Ejemplo 1: Hola Mundo

Los primeros programas que se confeccionaron fueron del tipo consola. Como en prácticamente todos los lenguajes de programación la primera aplicación que se crea es el “Hello World”. Aunque este es uno de los programas más sencillos que pueden crearse servirá para explicar la estructura básica de las aplicaciones en C#. Como sin duda ya se sabe, el programa Hello World al ser ejecutado entrega este mensaje en pantalla. El aspecto del código en C# de este programa es el siguiente:

```
using System;

namespace HolaMundo
{
    class Class1
    {
        static void Main(string[] args)
        {
            //
            // En este punto comienza la aplicación
            //
            Console.WriteLine("Hola Mundo");
            string a = Console.ReadLine();
        }
    }
}
```

El código de este programa, está casi totalmente proporcionado por el propio C#, que introduce por defecto los campos obligatorios que el programador deberá rellenar y complementar con algunos nuevos. La primera sentencia es del tipo using, e implica que podrán usarse todas las clases y objetos del espacio de nombres especificado, en este caso System. Console por ejemplo está contenido en este espacio de nombres. Mediante el comando using.System podemos invocar directamente a Console, ya que de otra manera habría que indicar el espacio de nombres al que pertenece en cada llamada (System.Console…). Al encontrarse este comando al principio del programa, afectará a todas las partes del mismo, ya sean dentro de métodos o clases.

La segunda sentencia, mediante el comando namespace, crea un nuevo espacio de nombres en el que ubicar el programa con el nombre HolaMundo. Todas las clases

declaradas en el programa estarán contenidas dentro de este espacio. Si se desea acceder a alguna de estas clases desde otro espacio de nombres externo a este, la sentencia deberá comenzar con HolaMundo, tal y como se ha explicado para el caso de System.

Una vez dentro del espacio de nombres correspondiente, es el momento de declarar las clases. Todo programa contendrá al menos una clase, en la que se encontrará el método principal de la aplicación, conocido en C# como método Main. En este caso, la clase creada a tal efecto tiene el nombre del Class1. Como ya se ha visto, si se desea acceder a un objeto o método de esta clase desde un punto externo al espacio actual, se deberá escribir la sentencia HolaMundo.Class1.NombreDelMétodo.

La clase recién declarada contiene un solo método. Este es el método Main, que contienen todos los programas de C# y que es donde comenzará la ejecución del programa. La línea de declaración del método tiene una estructura interesante. En primer lugar se encuentra el tipo del método, que en este caso es static. Este tipo se usa en declaraciones de clases y de métodos principalmente. La palabra static implica que todos los métodos contenidos en ella son estáticos (en caso de utilizarla en la declaración de una clase) o que el método no cambia aunque lo hagan los atributos de la clase u objeto al que pertenece (si se declara en un método). Uno de los inconvenientes de este tipo de declaración es que no se puede acceder a miembros estáticos de una clase aunque se haya instanciado con anterioridad. La siguiente palabra de la declaración es Void e indica el tipo de datos que devolverá el método. En este caso, el método no devuelve valor alguno y por eso el tipo es void (vacío en inglés). Después de esto se encuentra el nombre del método (Main) y entre paréntesis el tipo de datos que puede recibir. Si el paréntesis está vacío el método no aceptará ningún parámetro de entrada. En este caso aceptará un argumento de entrada del tipo matriz de cadenas de texto (String[]), que tiene por nombre args. Este parámetro se representa los argumentos de la línea de comandos, es decir, puede recibir mensajes escritos por el usuario.

Por último, en el interior del método se encuentra la aplicación propiamente dicha. Antes de comenzar con el código ejecutable, pueden verse unas líneas de comentarios, que se escriben tras dos barras (//). El contenido de las líneas que comienzan por estos dos caracteres no se ejecutará. En principio, todo lo que se necesita para realizar el programa HelloWorld es la línea: Console.WriteLine (“Hola Mundo”);. Esta línea invoca el método WriteLine de la clase Console (a su vez contenida en el espacio de nombre System, como se ha visto con anterioridad). Este método escribe el mensaje proporcionado entre paréntesis y comillas en una línea de la ventana de comandos y salta a la siguiente línea. El salto de línea está contenido en el método, e incluso existe otro similar que no lo incluye, llamado simplemente Write. Con esta única sentencia se lleva a cabo lo pretendido, sin embargo no se podrá visualizar el resultado, ya que al terminar la escritura se cerraría el programa impidiendo ver el mensaje. Para evitarlo puede incluirse otra sentencia: String a = Console.ReadLine( );. Mediante ella, se crea una variable del tipo cadena de texto llamada ***a*** y se asocia al valor que contenga la línea de comandos. Puesto que la línea está vacía, al encontrarse la posición del puntero una línea por debajo del mensaje, el programa esperará a que contenga algún valor, que habrá de ser introducido por el usuario. Esto permitirá visualizar el resultado y cuando se desee cerrar el programa solo se deberá pulsar cualquier tecla y validarla mediante el botón Enter del teclado.

Este es el ejemplo más sencillo de un programa en C#, pero ha servido para explicar brevemente las declaraciones de variables, espacios de nombres, clases y métodos, que constituyen la base del lenguaje.

3.3 Ejemplo 2: Llamando a métodos

Este es un nuevo programa de consola, que permitirá comprender como se realizan las instanciaciones. El objetivo de la aplicación será obtener dos números proporcionados por el usuario, sumarlos y presentar el resultado. El código es el siguiente:

```
using System;

namespace Aplicación1
{
    public class Metodos
    {
        public int Suma(int parametro1, int parametro2 )
        {
            int Resultado = parametro1 + parametro2;
            return Resultado;
        }
    }

    static class Program
    {
        static void Main(string[] args)
        {
            //
            // En este punto comienza la aplicación
            // Se piden los parámetro de la suma
            //
            Console.Write("Escriba parámetro 1: ");
            string stringpar1 = Console.ReadLine();
            Console.Write("Escriba parámetro 2: ");
            string stringpar2 = Console.ReadLine();
            //
            //Se convierten los parámetros de tipo string a int
            //
            int par1 = Convert.ToInt32(stringpar1);
            int par2 = Convert.ToInt32(stringpar2);
            //
            // Se instancia la clase y se llama al método
            //
            Metodos Inst = new Metodos();
            Resultadofinal = Inst.Suma (par1,par2);
            //
            //Se muestra el resultado
            //
            Console.WriteLine(Resultadofinal);
```

```
            Console.ReadKey();
        }
    }

}
```

Este es un programa bastante completo, por lo que habrá que analizarlo parte por parte. En primer lugar, se encuentra la sentencia using y la declaración del espacio de nombres, tal y como se hacía en el anterior programa.

En este caso el espacio de nombres contiene dos clases. Por un lado está la clase Program, que es la que contiene el método Main, y por otro la clase Metodos, que incluirá el método al que se invocará. La declaración de esta última clase (Metodos) es similar a la que se explicaba en el ejemplo anterior, sin embargo cambia la sentencia static por public. Esta sentencia corresponde a un modificador de acceso, y significa que no hay restricciones para acceder a la clase o método en cuestión.

El método contenido en esta clase lleva por nombre Suma y también es del tipo public. El formato del dato devuelto será de número entero (int) y se espera que reciba dos parámetros de entrada de este mismo formato, llamados parametro1 y parametro2. En lo que respecta a la ejecución del método, simplemente se crea una nueva variable de tipo entero con nombre Resultado y se le asigna el valor de la suma de los dos parámetros. La última sentencia del método corresponde a la necesidad de enviar el valor de la variable Resultado al programa principal.

Como ya se ha explicado, la clase Program contiene el método Main de la aplicación. La declaración de este método es igual a la que se ha explicado en el ejemplo anterior. El programa comenzará pidiendo al usuario que escriba el primer parámetro de la suma, y una vez leído este, mediante la sentencia Console.RealLine(), repetirá la operación para el segundo parámetro. Estos dos valores serán almacenados en sus variables correspondientes del tipo string, ya que todo lo que escribe el usuario en la consola de comandos tendrá formato de texto. Aunque el texto esté formado por números, no es posible sumar datos en este formato, por lo que habrá que implementar una conversión. El método que permite realizarla es Convert (que pertenece al espacio System). El resultado se guardará de nuevo en dos variables del tipo adecuado (número entero en este caso).

En este punto se tienen los valores a sumar. Para poder enviarlos al método que realiza la suma en primer lugar se ha de instanciar la clase. Para hacerlo, se implementa la sentencia: Metodos Inst = new Metodos ();. La palabra clave de la sentencia es new, ya que es la que indica que se creará una nueva instancia de la clase especificada. En este caso la palabra new está siendo utilizada como operador, no como modificador. Una vez se tiene la instancia de la clase, es posible invocar el método. Cuando se invoque no se debe olvidar proporcionar los parámetros para la suma entre paréntesis, con lo que la sentencia quedaría de la siguiente manera: Inst.Suma (par1, par2);. Dado que el método Suma devuelve un valor, la invocación no puede realizarse directamente, sino que se debe asociar a una nueva variable del formato adecuado. En este caso la variable es de tipo entero y lleva por nombre Resultadofinal. Esta variable recibirá el valor de la

variable privada Resultado, contenida en el método. Dado que esta variable ha sido declarada únicamente dentro de Suma, su nombre puede ser usado en el resto del programa considerándose una variable diferente. También es importante resaltar la importancia de las mayúsculas en el lenguaje, ya que C# diferencia entre mayúsculas y minúsculas y por lo tanto dos variables Inst e inst, serán consideradas diferentes.

Para finalizar, el programa muestra en pantalla el resultado de la operación. Como ocurriera en el ejemplo anterior, la última línea es la utilizada para que pueda visualizarse dicho resultado. En este caso se ha optado por el método ReadKey, ya que permite evitar el uso de una variable extra.

3.4 Ejemplo 3: Estructuras condicionales, bucles y Arrays de datos.

En C# también pueden implementarse las estructuras condicionales clásicas (if y else) y las estructuras iterativas (for, while, until…). Los arrays de datos también pueden utilizarse en C# y serán de gran utilidad para implementar todo tipo de aplicaciones. Un ejemplo de programa que utiliza las estructuras citadas sería el siguiente:

```
using System;

namespace Aplicación1
{
    static class Program
    {
        static void Main(string[] args)
        {
            int i = 0;

            Console.Write("Escriba número de entradas: ");
            string strnumpar = Console.ReadLine();
            int numpar = Convert.ToInt32(stringnumpar);

            string[] Parámetros;
            Parámetros = new string[numpar];

            for (i=0;i<=numpar;i++)
            {
                Console.Write (“Escriba entrada ”+i+”: ”);
                Parametros[i] = Console.ReadLine()
                if (Parámetros[i]== ““)
                {
                      Parámetros[i] = “Vacío”;
                }
            }

            for (i=0;i<=numpar;i++)
            {
                Console.WriteLine(“Entrada”+i+”: ”+Parametros[i]);
            }
```

```
            Console.ReadKey();
        }
    }

}
```

El objetivo de la aplicación es crear una lista con un número de entradas definido por el usuario y mostrarla al cuando todas las entradas hayan sido escritas. El comienzo del programa es igual a los anteriores por lo que no será necesaria una nueva explicación. El método Main comienza inicializando una variable de tipo entero llamada **i** con un valor de 0. Esta variable se utilizará como contador más adelante en el programa. Tras esto, se pide al usuario que especifique el número de entradas de la lista y una vez obtenido el string se convierte a número entero. Este es el momento en el que se crea el array (matriz). La creación de un array se lleva a cabo en dos pasos. En primer lugar se declara, con la sentencia string [] Parámetros, en la que consta el tipo de datos que contendrá y el nombre de la matriz. El segundo paso es la instanciación, en la que se asocia el nombre a un array de longitud determinada: Parametros = new string[i], donde i será la longitud del array. También pueden crearse matrices de dos, tres o más dimensiones, pero en este caso con una sola será suficiente. Una vez hecho esto ya pueden insertarse valores en el array.

Tras estos e encuentra el primer bucle for del programa. Este tipo de bucles son muy comunes en programación, por lo que no será necesario explicar su funcionamiento. La sintaxis de estos en C# es la siguiente: for (i=0 ; i<=5 ; i++). El primer parámetro de los tres que se encuentran dentro del paréntesis es la inicialización del contador. El segundo es la condición que ha de cumplir dicho contador para que se ejecute el bucle. El tercero indica que tras la ejecución de todas las sentencias del bucle, el valor del contador se incrementará en uno. Este último parámetro se conoce como postincremento. En su lugar se podrían colocar otros operadores como el postdecremento (i--), que indicaría que el contador reduce su valor en una unidad al terminar cada iteración, preincremento (++i), en el que el incremento se realiza antes de comenzar la ejecución del bucle, o por último el predecremento(--i), según el cual el valor se reduciría al comienzo de cada iteración. Como puede observarse, el uso de estos operadores otorga gran flexibilidad a los bucles.

Este primer bucle, es el encargado de leer las entradas escritas por el usuario y guardarlas en las posiciones correctas del array. Esta operación se lleva a cabo mediante la sentencia: Parámetros [i] = Console.RealLine( );. Dado que i es el contador, se irá modificando en cada iteración, posibilitando que no se escriba en ningún momento sobre una posición previamente utilizada. Además de esto, se ha añadido una pequeña sentencia condicional con el objetivo de llenar las entradas que no sean escritas por el usuario. La estructura de los condicionales if es la siguiente: if (Parámetros[i] == “”);. Si la condición entre paréntesis se cumple se ejecutarán las ordenes siguientes. Es importante resaltar el uso de la doble igualdad (==). En C#, un solo signo de igualdad (=), corresponde a la asignación de un valor, por ejemplo si se desea que la variable i tenga valor 3 se escribiría i=3. Sin embargo, si lo que se requiere es expresar una

condición de igualdad entre dos términos, como en el caso de las estructuras condicionales, se habrá de usar ese mismo símbolo dos veces. Para expresar desigualdad se utiliza la combinación de un signo de exclamación y un igual (!=).

Una vez se tienen todos los datos almacenados en el array solo queda mostrarlos ordenadamente. Para ello se utiliza el segundo bucle. De la misma manera en que se hacía en el anterior, en cada iteración se escribirá en pantalla uno de los datos con su correspondiente número de entrada, dando como resultado un lista de la forma: Entrada1 : Nombre1, Entrada2 : Nombre2, etc.

Como en los ejemplo anteriores también aquí se ha utilizado la última sentencia Console.ReadKey() para que sea posible la visualización del resultado. Con estos ejemplos, ya se tienen los conocimientos básicos para comprender el código del programa. Existen otras estructuras que se han usado, pero serán explicadas a su debido momento.

4 Solución adoptada

4.1 Bienvenida

4.1.1 Descripción

Es la clase que representa la ventana de bienvenida del programa. Es el punto principal del programa, ya que tanto la inicialización como la finalización del programa se llevan a cabo desde esta pantalla. Tiene el formato de una ventana de Windows ordinaria, con los botones de maximizar, minimizar y cerrar en la parte superior derecha de la misma. Además de estos, la ventana contiene dos botones con los rótulos ENTRAR y SALIR. La función de ambos es fácilmente deducible por sus nombres. El botón ENTRAR, inicializa un nuevo formulario (***Form3 Form3 = new Form3();***) y lo hace visible mediante el comando ***Form3.Visible = true;*** . El botón SALIR cierra el formulario mediante el comando ***this.Close();*** cerrando consigo la aplicación. Al pulsar el botón Entrar, la ventana actual queda en un segundo plano, de tal manera que si se cierra por error cualquiera de las ventanas de la aplicación no se cerrará el programa.

4.1.2 Propiedades

En este primer form solo se ha utilizado la propiedad *Visible* de la cale *Form.* En este caso, se configura con el valor *true* para que se muestre el nuevo formulario, pero en otros casos se le podría dar el valor *false* y mantener la aplicación funcionando sin necesidad de visualizar el formulario. En este caso, dado que el nuevo formulario corresponde al menú de la aplicación, conviene que sea visible.

4.1.3 Métodos

En Form1 tan solo se utilizan dos métodos. El primero, es común a todos los formularios, y viene implementado por el propio Visual Studio, es el método *Initializecomponent();*, que dibuja el formulario actual. El segundo, es el método *Close().* Como su nombre indica, se utiliza para cerrar el formulario. Este método, va asociado a la palabra clave *this.* Esta palabra, se utiliza para hacer referencia a la instancia actual de la clase, en este caso Form1(). Esta forma de proceder es muy útil cuando se requiere tener acceso a objetos o métodos de diferentes clases con nombres similares o cuando es necesario pasar un objeto como parámetro a otro método.

4.1.4 Aspecto final

El aspecto de la ventana de bienvenida a la aplicación es el siguiente:

4.2 Menú Principal

4.2.1 Descripción

Es la clase que representa la ventana del menú principal de la aplicación y desde ella se puede acceder a todas las funciones de la misma. Esta ventana consta de seis botones rotulados de la siguiente manera: CONFIGURACIÓN, CAPTURA, GRÁFICOS, VIDEO, AYUDA y SALIR. Cada uno abre la parte correspondiente del programa. Pulsando en el primer botón se accede a una ventana en la que se puede visualizar y cambiar la configuración de todos los módulos y sensores de la instalación. Pulsar el botón de Captura da acceso a un nuevo formulario en el que puede activarse la captación de datos de los sensores y visualizarlos en tiempo real, así como configurar los intervalos de captura. El botón Gráficos abre una ventana en la que se pueden observar históricos de los datos capturados por la aplicación en forma de gráficas lineales. El botón de Video permite acceder a una interfaz web y conectarse a una cámara emitiendo en live streaming. Ayuda redirige al usuario a una ventana en la que se encuentran las instrucciones de uso del programa. El sexto y último botón (Salir) finaliza la ventana de menú, dejando ver la ventana de inicio.

4.2.2 Propiedades

Se utiliza la propiedad *Visible*, de la misma manera que en el Form1.

4.2.3 Métodos

Se utilizan los mismos que en Form1 ya que la operatividad es la misma en ambos formularios.

4.2.4 Aspecto final

El aspecto del menú principal de la aplicación es el siguiente:

4.3 Configuración

4.3.1 Descripción

Es la ventana de configuración de los módulos que conforman la instalación. Desde ella es posible visualizar la configuración actual y modificar los parámetros básicos de la misma. Consta de una primera sección, situada en la parte superior de la ventana, que permite seleccionar el módulo y la entrada que se quiere visualizar mediante un control numérico y un cuadro de texto desplegable. Dependiendo del módulo que se seleccione el cuadro mostrará las entradas asociadas al mismo. En la parte superior derecha se pueden encontrar los botones que inician los procesos de actualización o descarga de la configuración actual del módulo. Las funciones específicas de cada uno se explicarán a continuación. La última sección es la más extensa y consta de una serie de cuadros de texto en los que se presentará la información sobre la configuración y desde la que se podrán modificar dichos datos. Esta información podría ser subdividida en dos partes, por un lado la información que puede modificarse y por otro lado la que solo se puede visualizar. Se ha tomado la decisión de diferenciar entre estos dos tipos debido a que existen algunos datos que no son de fácil comprensión a nivel de usuario y que sin embargo ofrecen información valiosa para alguien especializado. Los parámetros configurables son el nombre de la entrada, la localización del módulo al que pertenece, el tipo de entrada, que puede ser analógica, digital o salida. Uno de los parámetros más importantes de este tipo son las unidades de los datos recibidos, que pueden ser voltios, amperios o resistencia (que se ha preconfigurado como grados, ya que se utiliza para medición de temperaturas). Los últimos parámetros configurables son los límites de medición del sensor, que son importantes para la correcta interpretación de los valores, y la precisión de los datos (el número de cifras decimales con el que serán representados.

Los parámetros no configurables se ha añadido como una sección libre que pueda ser configurada especialmente para cada aplicación. En este caso, se ha decidido añadir el formato de los datos (Float, usint16, sint32…), que será útil para cualquiera que desee ahondar en la comunicación entre la aplicación y los módulos, y la fecha y hora de la última actualización, que también puede ser de interés.

4.3.2 Diseño

4.3.2.1 Controles y cuadros de texto

Antes de comenzar con la descripción de los eventos asociados a esta ventana, es necesario, como suele ser habitual crear algunas variables del tipo string o de tipo entero, que servirán de apoyo para realizar las funciones necesarias, como podrían ser contadores etc.

El primer evento que se puede encontrar en el código está asociado al control de tipo numérico de la parte superior de la ventana, llamado ***numericUpDown1*** y rotulado como Módulo. La finalidad de este proceso es la de cargar. El evento en cuestión

comienza a ejecutarse al detectar un cambio en su valor, inducido por el usuario. La finalidad de este evento es que se muestren las entradas asociadas a cada módulo cuando este es seleccionado. El proceso comienza almacenando el número contenido en el control en una variable de tipo entero llamada i. Para recoger el valor del control, se ha de utilizar la sentencia numericUpDown.Value que representa dicho número con formato de número decimal. Dado que lo números siempre serán enteros y su conversión a otros formatos puede hacerse en muchas ocasiones de forma implícita, se ha optado por convertir este número decimal en entero. Para ello se utiliza la siguiente nomenclatura: ***i = decimal.ToInt32(numericUpDown1.Value)***. Para evitar que se acumulen datos en el combobox correspondiente a los nombres de las entradas, es necesario borrar todo su contenido. Para ello se utiliza el método clear de la propiedad Items del combobox. En caso de que sea la primera vez que se cargan los nombres en el control, no habrá problema alguno, sin embargo si se ha hecho una actualización o una descarga de la configuración existente en los módulos, podría aparecer este problema.

Para cargar los nombres de las entradas en el control se utilizará un método como parte del proceso. Este método, llamado *nombres*, devolverá un string en el que podrán encontrarse cada uno de los nombres de las entradas. El método sigue el procedimiento siguiente: En primer lugar guarda en variables locales los datos que se le pasan desde el programa. Esto no es estrictamente necesario, pero facilita la comprensión de lo que contiene la variable y a la vez permite no tener que escribir el nombre (de una longitud considerable) una y otra vez. Los datos que se introducen en el método son dos. El primero es el índice del módulo del que se quiere conocer las entradas y el segundo es lo que se ha dado en llamar palabra clave, que es la nomenclatura con la que aparece el dato que se desea recuperar en el archivo. En caso de los nombres la palabra clave es *Na*. Una vez se tienen almacenados estos datos, se comprueba si el archivo de configuración del módulo al que se hace referencia existe. En caso de que así sea se comienza lee y almacena en una variable de tipo streamreader. Se va leyendo el texto almacenado línea por línea hasta que termina mediante un bucle while. Para cada línea se le aplica un proceso de separación de cada palabra, que se almacena en una nueva variable. La sentencia que permite esto es: ***string[] split = line.Split(new Char[] { ')', ' ', ',', '.', ':', '=', '<', '>' });*** . El programa al encontrar cualquiera de los caracteres especificados realizará un corte en el string. Como se sabe que la palabra clave siempre es la primera palabra de la línea, no hay más que comparar esa palabra inicial de cada línea con dicha palabra clave. En caso de que coincidan, se podrá asegurar que la siguiente palabra de la línea es el nombre de una de las entradas. Este nombre se almacena en un string seguido de un carácter de separación predefinido, que en este caso es una barra. Aunque algo rudimentario, este proceso es el más adecuado para la aplicación ya que no se permite la devolución al programa principal de varias posiciones de un array de strings, que sería la opción más práctica, y por lo tanto no sería posible mostrar todos los nombres con una sola llamada al método.

Después de llamar al método y que este devuelva el string con los nombres de las entradas, se pasará a introducirlas en el combobox. Para ello, será necesario dividir el string de los nombres de la misma manera que se hacia en el método, valiéndose del carácter de separación. Una vez dividido el string guardaremos cada nombre en una posición de un array. Mediante un bucle for y utilizando el método add perteneciente ala

propiedad ítems del combobox se irán añadiendo nombres al listado. Para actualizar el combobox se utilizará la siguiente sentencia: ***this.Controls.Add(this.comboBox1)***;.

Una vez hecho esto, ya se podrán visualizar todas las entradas. Pero el proceso sigue ya que se ha decidido que sería de ayuda contar también con un cuadro en el que se especifique en que tipo de módulo están contenidas estas entradas. Se dispone de tres tipos de módulos distintos. El primero de ellos es el e.reader, que es el que centra la inteligencia de la instalación. El segundo tipo es el e.bloxx, que lo conforman distintas entradas/salidas configurables. Dado que estas entradas/salidas pueden contener variables virtuales y otros datos, se considerarán cada una de las entradas del e.bloxx como un módulo distinto. Por último, el tercer tipo de módulo es el e.bloxx digital, que consta de cuatro salidas digitales también configurables. En el archivo de configuración, sin embargo, no aparecen con la nomenclatura de e.reader o e.bloxx, sino con un código numérico. El código para el e.reader es el 230, el e.bloxx viene representado por un 200 y el e.bloxx digital por un 216. Para detectar el tipo de módulo, se llevará a cabo un proceso muy similar al que se aplicaba para los nombres, de hecho, se utilizará el mismo método que en el caso anterior, con la diferencia de que esta vez la palabra clave será ModTy (abreviatura de Module Type). Para comprobar si la entrada pertenece a uno u otro módulo, se deberá comparar y mediante un If evaluar si la condición se cumple y escribir el nombre del módulo en el cuadro correspondiente. Si no se encuentra la información sobre el tipo de módulo en el archivo o no hay ninguna entrada seleccionada, se mostrará el mensaje N/A en el cuadro de texto.

Una vez se han cargado los nombres de las entradas en el control de tipo lista, se puede proceder a elegir una de ellas. Al hacer esto, se acciona un nuevo evento con el cual se cargarán los parámetros de la configuración en los cuadros de texto correspondientes. El primer cuadro de texto a rellenar es el del nombre, que simplemente se transfiere del control de lista al cuadro. El segundo parámetro es la localización. Para recuperar este dato del archivo de configuración se utilizará el método *reconocimiento* de la clase *guardarnombres*. Este método es muy similar al utilizado para la recuperación de los nombres de las entradas que se ha analizado anteriormente. La palabra clave con la que se buscará el dato en el archivo es *Loc*, como se puede ver en la llamada al método: ***eltexto2 = localizacion.reconocimiento(i, "Loc");*** . Para evitar problemas, es preferible que tanto la localización como los nombres de las entradas no contengan espacios ni barras del tipo “/”. Si es necesario que estos datos consten de dos palabras puede hacerse la separación mediante una barra baja o un guión.

El siguiente cuadro a cumplimentar es el del tipo de entrada. En este caso no se trata de un cuadro de texto sino de un control de tipo lista. Esto es así debido a que el contenido del mismo no es libre, como pudiera ser la localización o el nombre de la entrada, sino que debe ajustarse a unas normas. Los tipos de entradas/salidas contemplados en el e.reader son: Entrada analógica, entrada digital, salida digital, entrada/salida aritmética y entrada/salida de alarma. Para esta aplicación se ha decidido simplificar los tipos a los tres más comunes, que son las entradas analógicas y las entradas y salidas digitales. Esta simplificación se lleva a cabo por el hecho de que no se dispone del material necesario para realizar las pruebas de funcionamiento para los tipos de entradas/salidas omitidos. Para la recuperación de los datos, se utiliza de nuevo el método reconocimiento, sustituyendo la palabra clave por Ty. Una vez hecho esto, se obtienen unos números del

1 al 6 omitiendo el 5. Cada número corresponde a un tipo de entrada. No se especifica una clave pero con breve análisis se puede concluir que el número 1 corresponde a una entrada analógica, el 2 a una entrada/salida de tipo aritmético, el 3 representa una salida digital, el 4 se utiliza en entradas digitales, el 5 no aparece en ningún caso y el 6 corresponde a entradas/salidas de alarma. Una vez conocido esto, solo hay que asociar cada número recuperado con la posición del tipo de entrada correspondiente en el listado y seleccionarlo.

El siguiente cuadro también es de tipo lista y en él se representarán las unidades del valor entregado por el sensor en la entrada seleccionada. Este caso es muy similar al anterior ya que los datos recuperados tampoco serán representados directamente. La palabra clave para la búsqueda de datos de este tipo de DaUn. Los valores recuperados por el método pueden ser de varias formas aunque todos serán similares. En primer lugar, es importante señalar cuales son las unidades que el e.reader es capaz de interpretar, a saber: Voltaje, resistencia (cuyo principal uso es la medición de temperaturas) o corriente (habitualmente en miliamperios). En el archivo de configuración el voltaje puede venir representado de dos maneras “V” y “_V”. Como se puede observar son prácticamente iguales, sin embargo para una aplicación informática no lo son. Los datos de corriente suelen representarse con la siguiente palabra “mA”, que obviamente se refiere a miliamperios. El caso de la resistencia es el más complejo, ya que no viene representada por sus unidades habituales, que son los ohmios. En lugar de esto, el e.reader, conociendo que la aplicación de la resistencia es la medición de temperatura, convierte automáticamente los ohmios a grados Celsius. Por lo tanto la resistencia puede ser representada de las formas “ºC” y “_ºC”. En alguna ocasión ha sucedido que el procesador de texto del programa no reconocía el símbolo de grado, pasando las palabras anteriores a la siguiente forma “ C” y “_ C”.

A continuación, pueden encontrarse dos cuadros en los que se podrán configurar el rango del sensor. En el primero de los cuadros se situará el máximo de la variable y en el segundo el mínimo. Este es un primer paso hacia la implementación de alarmas ya que en ellos el usuario mostrará cuales son los valores máximos y mínimos deseables para cada entrada. Además, ayudarán a una mejor comprensión de los datos por parte del usuario cuando se analice la variación de datos en el tiempo ya que aportará dos valores de referencia. Estos dos parámetros han de ser configurados por el usuario, en caso de que no sea así, estos están preconfigurados para mantenerse en el máximo y el mínimo del rango del sensor. De todas formas, si no se desea hacer uso de esta función se recomienda que se sitúen ambos valores en cero para que no entorpezcan la visualización. El proceso de extracción de los datos es idéntico al del resto de parámetros, sin embargo, en el caso de los máximos y mínimos existe una peculiaridad, y es que si el campo de las unidades no muestra ningún valor no habrá límites, como es obvio. Esta circunstancia se da por ejemplo en el caso de las salidas digitales, para las que no se especifican unidades. Por lo tanto previamente a la llamada al método reconocimiento, se ha de evaluar (mediante una sentencia if) si existe información sobre las unidades. En caso de que el campo esté vacío o contenga valores no válidos (tales como barras, puntos o guiones) no se realizará la llamada al método y por tanto se escribirá en los cuadros el texto *N/A*.

El siguiente parámetro es la precisión del dato capturado por la entrada. Por precisión se entiende el número de dígitos decimales con los que será representado el valor. Por lo tanto, el número 5.17 tendría una precisión de 2 o el número 28.361 la tendrá de 3. La palabra clave de este parámetro es *DaPr*. Con este, terminan los parámetros configurables, los que siguen serán meramente informativos y no se podrán modificar desde la interfaz.

Como datos no configurables, se han incluido los dos que se han considerado más interesantes, aunque podrían haberse incluido otros. El primero de los parámetros es el tipo de dato, que representa el formato del dato que se recibirá en la entrada. Como norma general será de tipo Float (coma flotante), sin embargo también puede tomar otros valores como unsint (número entero sin signo), bolean (que puede tomar valores verdadero y falso) o double (numero real de precisión doble). El segundo y último parámetro ha sido rotulado como Última Act., y se refiere a la fecha y hora de la última actualización. Esta fecha se graba al realizar la comprobación de que la actualización ha sido correcta (al pulsar el botón descargar), por lo que será igual para todas las entradas.

4.3.2.2 Botones

El siguiente paso en el programa es la definición de los eventos asociados a los botones. Como se ha explicado antes, la ventana cuenta con cuatro botones, a saber: Actualizar, Descargar, Ayuda y Menú. Cada uno de ellos llevará a cabo una función distinta, las cuales se irán especificando a continuación.

4.3.2.2.1 Menú

El primer botón del que se hablará será el botón Menú. Su función es actuar de acceso al menú principal, es decir un botón de salida. Las sentencias son análogas a las que se pueden encontrar en el botón de salida del menú principal y el resto de la aplicación. Como una ayuda visual, se ha decidido distinguir estos botones con un color naranja en sus letras para que no haya confusiones.

4.3.2.2.2 Ayuda

El siguiente botón es el de ayuda. Su función es dirigir al usuario a la ventana de ayuda, donde podrá encontrar las soluciones a las dudas más habituales sobre el funcionamiento de la aplicación así como explicaciones sobre las funciones que sea posible realizar a través de la aplicación. La ventana de ayuda se abrirá superpuesta al formulario actual, por lo que no se perderán los cambios no guardados. Tanto este botón como el anterior, son comunes al resto de ventanas de la aplicación.

Los dos botones restantes (Actualizar y Descargar) son únicos de esta ventana y llevan a cabo las funciones más importantes del formulario. Por simplicidad y cronología, se comenzará la explicación por el botón Descargar.

4.3.2.2.3 Descargar

El botón descargar lleva a cabo 2 funciones distintas. En primer lugar descarga los archivos de configuración que están cargados en el e.reader, una vez hecho esto crea un archivo en el que especifica los nombres de las entradas y el módulo al que pertenecen (este archivo tendrá varios usos a lo largo del programa).

La primera acción que se realiza al ejecutarse el evento es poner el contador de número de módulo en la posición uno, para que el barrido comience desde el primer módulo. Una vez hecho esto, se llama al método *descargar* de la clase guardarnombres. Esta llamada se hace de la forma clásica por lo que no se entrará en mayores detalles.

El método descargar comienza conectándose al e.reader mediante protocolo FTP. Para ello, se cuenta con una biblioteca de clases llamada FTPFactory, de la cual, debido a su extensión, se explicarán los métodos más relevantes en su propia sección de este documento. Para realizar la conexión hacen falta varios pasos que se harán automáticamente, por lo que serán invisibles al usuario. En primer lugar, se ha de especificar la IP del servidor al que se desea conectarse. En este caso el servidor es el e.reader. Una vez se comience la conexión el módulo requerirá un nombre de usuario y una contraseña por lo que habrá que especificarlas también previamente. Una vez se han enviado estos tres datos a la biblioteca, se puede proceder a la conexión mediante el método *login* de la misma clase. Una vez el módulo acepte los datos enviados ya se puede comenzar a operar con él.

La primera operación que se lleva a cabo es la descarga de un listado con los nombres de todos los archivos contenidos en el servidor. Para almacenar el listado, se crea un archivo de texto en el que se escribirán los nombres. Una vez hecho esto, se procederá a descargar los archivos de configuración. Para ello, no hay más que crear un bucle que vaya comprobando cuales de los archivos de configuración se encuentran almacenados en el servidor y descargarlos. Para hacer la comprobación se utilizará una sentencia condicional tipo *if*. La descarga se hará utilizando el método *download* de la clase *FTPFactory*. El método descargar no devuelve ningún parámetro. El tiempo de espera aproximado para realizar todas las descargas (con 10 archivos de configuración) es de tan solo un minuto, lo cual mejora ampliamente el tiempo que se invierte para tal fin con el software de Gantner. En este punto ya se ha completado la primera de las funciones.

La segunda función del botón Descargar es la de crear un documento en el que se especificará cada nombre de entrada y el número del módulo al que corresponde, así como la información de máximos y mínimos para futura implementación de alarmas. Para ello se crea una instancia de la clase XmlWriter. Los archivos Xml están escritos según el estándar que les da nombre. Su principal uso se da en Internet ya que en este tipo de aplicación es necesario que la información se intercambie en formatos compatibles. El principal cometido de este estándar no es más que el de estructurar la información dentro de los archivos de manera que pueda ser leído desde el mayor número de plataformas posible. El nombre del que se dotará al archivo será entconfig.xml. En la primera línea de información del archivo se escribirá la fecha y

hora actuales. En las líneas siguientes, se irán escribiendo los nombres de las entradas, los números de módulo y los máximos y mínimos con ayuda de dos bucles, unos for y otro de tipo while. Las sentencias serán de la siguiente manera:

```
while (comboBox1.Items.Count >= 1)
{
    for (k = 0; k < comboBox1.Items.Count; k++)
    {
        comboBox1.SelectedIndex = k;
        writer.WriteName("Entrada" + contador);
        writer.WriteElementString("e" + contador + "_Modulo", numericUpDown1.Value.ToString());
        writer.WriteElementString("e" + contador + "_Name", textBox2.Text);
        writer.WriteElementString("e" + contador + "_Max", textBox4.Text);
        writer.WriteElementString("e" + contador + "_Min", textBox5.Text);
        contador++;
    }
    numericUpDown1.Value++;
}
```

La información de cada entrada viene precedida de una etiqueta con el número de entrada al que corresponde tal información. Una vez finalicen todas la iteraciones del bucle se cerrará la escritura del archivo mediante un elemento conocido como EndElement. Todos los archivos de este tipo se cierran de esta manera. Además de esto, también se puede cerrar la instancia al escritos de archivos Xml mediante el método Close pero en este caso no será necesario ya que el a continuación se cerrará el evento, lo que a su vez cerrará todos los procesos derivados de él.

4.3.2.2.4 Actualizar

El último botón de la ventana es el de la actualización. Su función es capturar los datos que hayan sido modificados por el usuario, sustituir los antiguos valores por los nuevos y enviar el archivo de configuración modificado al módulo de tal manera que este sea capaz de interpretarlo correctamente. Para llevar a cabo este proceso se utilizará el método *actualización* de la clase *guardarnombres*. El método no devolverá ningún valor, pero necesita que se le proporcionen una serie de datos. En primer lugar necesita conocer el número del módulo que se desea actualizar para acceder al archivo correcto. También es necesario proporcionarle el índice de la entrada seleccionada en el control de tipo lista, es decir, la posición numérica de la entrada dentro del archivo (primera, segunda, quinta, etc.). El tercer dato necesario es la palabra clave, ya que es lo que determina el dato que debe buscar. El último dato es el nuevo valor que se desea dar al parámetro. Como puede observarse por la cantidad y naturaleza de los datos que han de enviarse al método, la actualización se hará parámetro por parámetro.

El primer paso del método es generar el nombre del archivo de configuración que se debe actualizar utilizando para ello el número de módulo proporcionado en la llamada. Utilizando la clase streamreader y un bucle while se va leyendo el archivo línea por línea y dividiendo cada línea en palabras con la ayuda del método split. Esta parte del proceso es igual a la que se llevaba a cabo en el método de reconocimiento. Una vez se tienen las líneas divididas se compara la primera palabra de cada línea para ver si

coincide con la palabra clave. En caso de que coincida, se introduce una nueva condición, por la que el índice de la entrada debe coincidir con el de un contador que va incrementando su valor cada vez que se detecta una aparición de la palabra clave. Con esta segunda condición se consigue que pese a que en un mismo archivo existan varias coincidencias con la palabra clave, el nuevo dato se escriba en la entrada a la que corresponde. Si ambas condiciones se cumplen se sustituye la línea completa por una nueva de la siguiente manera:

```
if (split[0] == palabraclave)
{
    if (cont == indiceentrada)
    {
        line = palabraclave+"="+nuevovalor;
    }
    cont = cont + 1;
}
```

La variable cont es el contador de apariciones de la palabra clave. Al final de la iteración se incrementa en uno ya que la palabra clave ha sido encontrada. Como se puede ver la nueva línea sigue la misma estructura de la antigua simplemente insertando el nuevo valor el la tercera posición tras la palabra clave y un signo de igualdad. Para escribir la nueva línea en el archivo habrá de utilizarse la clase streamwriter. La escritura se hará línea por línea, copiando cada una de ellas y tan solo cambiando la línea actualizada. El archivo resultante tendrá el nombre provisional de texto2.txt. Una vez terminada la escritura de este archivo se eliminará el archivo de configuración original y se renombrará texto2 para que coincida con el nombre adecuado. Una vez terminados estos pasos ya se tendrá el archivo de configuración actualizado y listo para ser enviado, sin embargo, existe una peculiaridad en la comunicación que hace que si se envía únicamente este archivo el módulo no lo leerá.

La solución se encuentra en un archivo adicional llamado #command.sta que ha de ser enviado junto con la nueva configuración. En principio es fácil pasar por alto este archivo debido a que no se almacena en el servidor, sino que es leído y descartado automáticamente. Gracias a un examen minucioso de la comunicación mediante wireshark fue posible descubrir su funcionamiento. En este archivo se ha de escribir el número del módulo que se desea actualizar, por lo que habrá que crear un archivo por cada módulo. Para ello se utilizará la clase streamwriter, al igual que se hacía en el caso del archivo de configuración. Una vez creados los dos archivos el método llega a su fin, devolviendo la ejecución al programa.

Una vez analizado el método al que se recurrirá en repetidas ocasiones se podrá explicar la ejecución del programa. En primer lugar se actualizarán los parámetros más sencillos, que son los que se representaban en los cuadros de texto, y que como se ha visto, no necesitaban de ningún procesamiento. Se llamará al método en cada ocasión enviando una palabra clave y el dato correspondiente para cada parámetro. El nombre, la localización, la precisión, el máximo y el mínimo serán los datos actualizados de esta

manera, quedando la actualización del tipo de entrada y de las unidades como casos especiales.

Para actualizar el tipo de entrada, es necesario conocer que tipo corresponde a que índice del control de tipo lista en el que están almacenados. La manera más sencilla de llevar a cabo esta operación es utilizando cláusulas if anidadas de la siguiente forma:

```
if (comboBox3.SelectedIndex == 0)
{
    tipo = "1";
}
else
{
    if (comboBox3.SelectedIndex == 1)
    {
        tipo = "4";
    }
    else
    {
        if (comboBox3.SelectedIndex == 2)
        {
            tipo = "3";
        }
        else
        {
            tipo = "";
        }
    }
}
```

El valor que el módulo espera encontrar se va escribiendo en la variable *tipo*, que posteriormente se enviará al método junto con el resto de parámetros. Si ninguno de los índices de la lista está seleccionado se le da un valor vacío a la variable.

La actualización de las unidades es la más compleja, debido a que el parámetro que debe sustituirse no es único. Para que el cambio de unidades sea efectivo se han de modificar además de DaUn, los parámetros AInMo, AInMoPar0, SeNa, LPCnt, X0, Y0, X1 e Y1. DaUn, como se ha comprobado anteriormente, contiene el valor de las unidades. SeNa muestra el nombre técnico del sensor, o más bien del tipo de sensor. LPCnt indica el número de puntos para el escalado de los valores y los parámetros X0, Y0, X1 e Y1 muestran los valores de dichos puntos. AInMo y AInMoPar0 son los dos parámetros que realmente representan las unidades del sensor, y lo hacen mediante un dígito entre el 0 y el 3. La combinación de estos dos valores es leída e interpretada por el módulo como si de un valor de unidades se tratara. Cabe destacar que estos dos valores solo aparecen para entradas analógicas, como se puede deducir de sus nombres (AIn como abreviatura de Analog Input). En caso de entradas digitales AInMo se sustituiría por DInMo y en caso de salidas digitales por DOutMo. Para alarmas y entradas/salidas de tipo aritmético no aparecen estos parámetros, como tampoco aparecen DaUn y SeNa.

Dado que el valor de las unidades también se elige de un control de tipo lista, se utilizará la misma estrategia de cláusulas if anidadas. Dentro de cada condición se actualizarán los valores que se han especificado en el párrafo anterior a excepción de DaUn, que se actualizará una vez terminada la ejecución de los bucles. La actualización se ha estructurado de esta manera para evitar problemas en caso de que el valor de las unidades esté vacío, como ocurría con las salidas digitales.

Los valores de los parámetros que corresponden a cada valor de unidades han sido determinados mediante observación de los archivos de configuración tras diferentes actualizaciones. En la tabla siguiente pueden verse los resultados para los valores de unidades habituales:

	DaUn	AInMo	AInMoPar	SeNa	LPCnt	X0	Y0	X1	Y1
Res. ºC	ºC	2	3	Pt_100	2	100	0	3e2	700
Voltaje	V	0	1	Voltage	2	-10	-10	10	10
Corriente	_mA	1	0	Current	2	4E-3	4E-3	2E-2	2E-2

Para los valores de temperatura, en realidad se utiliza una escala no lineal por lo que LPCnt suele estar entorno a 64, sin embargo, para valores centrales de temperatura, entre -20ºC y +80ºC puede aproximarse como una función lineal, lo cual hace que pueda simplificarse en gran medida su configuración ya que de otro modo deberían actualizarse las coordenadas de los 64 puntos. Por lo tanto, todas las cláusulas if tendrán una forma similar, adaptada claro está a cada valor de unidades. En este caso se presenta para voltaje:

```
if (comboBox2.SelectedIndex == 1)
{
    uni = "V";
    string u1 = "0";
    string u2 = "1";
    string u3 = "Voltage";
    string u4 = "2";
    string x0 = "-10";
    string y0 = "-10";
    string x1 = "10";
    string y1 = "10";
    guardarnombres actualV = new guardarnombres();
    actualV.actualizacion(nuival, comboBox1.SelectedIndex, "AInMo", u1);
    actualV.actualizacion(nuival, comboBox1.SelectedIndex, "AInMoPar0", u2);
    actualV.actualizacion(nuival, comboBox1.SelectedIndex, "SeNa", u3);
    actualV.actualizacion(nuival, comboBox1.SelectedIndex, "LPCnt", u4);
    actualV.actualizacion(nuival, comboBox1.SelectedIndex, "X0", x0);
    actualV.actualizacion(nuival, comboBox1.SelectedIndex, "Y0", y0);
    actualV.actualizacion(nuival, comboBox1.SelectedIndex, "X1", x1);
    actualV.actualizacion(nuival, comboBox1.SelectedIndex, "Y1", y1);
}
```

Como puede observarse, en primer lugar se crean las variables. Una vez se tienen todas se instancia la clase *guardarnombres* y por último se llama al método *actualización* una vez por cada dato que se deba actualizar en el archivo. Este proceso reescribirá dicho archivo ocho veces.

Una vez se ha terminado la ejecución del if correspondiente es el momento de completar la actualización. Para ello, se comienza reescribiendo el archivo de actualización una vez más, ya que no se había escrito el valor de DaUn. Una vez se conoce el nombre del archivo que se enviará al servidor (simplemente insertando el número de módulo que se va a actualizar en el lugar adecuado del nombre), llegará el momento de la conexión. El proceso seguido para conectarse al módulo en el método *descargar* es igualmente válido para este caso, por lo que se aplicará directamente. Para subir tanto el archivo de configuración como el archivo *#command.sta* al servidor se utilizará el método *upload* de la clase *FTPFactory*. Opcionalmente, pueden descargarse ciertos archivos de control de errores tales como #event.sta o #actual.sta. Se ha observado que si las subidas de ambos archivos se lleva a cabo sin un pequeño espacio de tiempo entre ellas pueden darse errores en la escritura de la configuración. Esto es debido a que el módulo necesita de unos segundos para que una configuración nueva se estabilice. Para evitar esto pueden descargarse alguno de los archivos de control de errores previamente citados entre los dos envíos, con lo cual se le dará el tiempo necesario al módulo para realizar los procesos necesarios.

4.3.2.3 Carga de imágenes y aspecto final

El último evento de esta ventana está asociado al control de tipo numérico con el que se eligen los módulos. Como se ha explicado previamente, dependiendo del tipo de módulo que esté seleccionado se índica un nombre en el cuadrote texto que se encuentra bajo el control. Para facilitar aún más el reconocimiento visual de los módulos se ha decidido añadir un cuadro de dibujo donde se mostrará la foto de dicho módulo. Esta última acción es meramente estética, pero a la vez hace que la elección de los módulos resulte más intuitiva. El aspecto final de la ventana de configuración es el siguiente:

4.4 Captura

4.4.1 Descripción

Es la clase que representa la ventana de Captura de la aplicación. Desde este formulario se visualizan y almacenan en tiempo real los datos obtenidos por los sensores de la instalación. La ventana consta de un cuadro de texto en el que escribir el intervalo de tiempo entre capturas que está situado en la parte superior, debajo de este se encuentra un control numérico que permite visualizar las entradas correspondientes al módulo que se elija mediante el mismo. Ambos controles pueden dejarse en blanco ya que se ha previsto que si esto ocurre el intervalo de captura se configurará por defecto a 10 minutos y se mostrarán todas las entradas. De esta manera, la aplicación se hace más intuitiva y se minimiza el efecto de algún posible olvido. En la parte central de la ventana se encuentra el cuadro de texto en el que se mostrarán los datos capturados. En la parte inferior de la ventana se ha colocado un pequeño cuadro de texto, en el que visualizar una sola entrada para casos en los que resulte incómoda la vista en forma de listado. Para visualizar una entrada en este cajetín, no hay más que seleccionarla en la lista del cuadro superior. En la esquina superior derecha de la ventana se encuentran tres botones, con los rótulos Capturar, Stop y Volver a Menú. El primero de ellos inicia la obtención, almacenamiento y visualización de los datos, el segundo detiene el proceso de captura y el último devuelve al usuario al menú principal.

4.4.2 Diseño

En primer lugar se crea una instancia de la clase *FTPFactory*. *FTPFactory* contiene los métodos necesarios para la conexión vía FTP con un servidor. La nueva instancia se almacena en una variable que en este caso tiene como nombre *ff*. De esta manera, para llamar a un método de la clase citada lo haremos de la manera *ff.nombremétodo(x)*, donde *nombremétodo* es el nombre del método al que se quiere acceder y x son los parámetros que se le pasan a dicho método. Una vez creada la instancia, inicializamos unas variables de tipo *string* e *int* que utilizaremos posteriormente.

Este formulario, como la mayoría de los formularios de este tipo, se estructura en forma de eventos. Un evento es el resultado de una acción, ya sea llevada a cabo por el usuario, como sería el caso de pulsar uno de los botones o escribir en un cuadro de texto, como si la acción se produce de forma automática. Más adelante se muestran un par de ejemplos de acciones automáticas.

El primer evento que se encuentra al revisar el programa es el de la pulsación del botón *Volver a menú*. El constructor de dicho evento es el siguiente: ***private void button3_Click(object sender, EventArgs e).*** Si se analiza la frase se entenderá mejor el funcionamiento del mismo. La palabra private indica que el evento solo es accesible desde la clase actual, es decir Form4. Esto tiene sentido ya que el botón está situado en esta ventana y sería ilógico poder llamarlo desde otras partes de la aplicación. Por lo

general todos los eventos asociados a pulsadores serán del tipo *private.* La siguiente palabra indica el tipo de dato que devuelve el evento. En este caso, el evento no retorna valor alguno, por lo que este indicador se muestra de tipo *void* (vacío). Dependiendo del método o evento, esta palabra podría sustituirse con valores como *string*(palabras), *int*(números enteros), *bool*(operadores booleanos como true o false), etc. Tras las declaraciones del tipo se encuentra el nombre del evento, en este caso *button3_click*. Este nombre consta de dos partes, como puede verse, primeramente se identifica el objeto que genera el evento (el tercer botón) y en segundo lugar se especifica la causa del evento (hacer clic sobre dicho botón). La parte de la sentencia entre paréntesis proporciona información adicional al programa. La sentencia *object sender* indica que fue lo que provocó el evento y la sentencia *EventArgs e* detalla otros parámetros sobre el evento, pero esto no es excesivamente relevante ya que se trata de información que afecta al funcionamiento interno del lenguaje de programación. La activación del evento de clic en el botón 3 de la ventana, transporta a una rutina específica, que en este caso abre la ventana de menú (clase Form3) y cierra la ventana actual. Para llevar a cabo esta operación utiliza las sentencias ***Form3.Visible = true;*** y ***this.Close();*** que ya han sido discutidas previamente en este documento.

Más adelante, se encuentra un nuevo evento del tipo *click*, esta vez asociado al botón 2, que recordemos estaba rotulado como Stop. El constructor es muy similar al que se acaba de explicar, por lo que no será necesario entrar en más detalles sobre él, simplemente lo mostraremos. Es el siguiente: ***private void button2_Click(object sender, EventArgs e).*** La rutina de este evento es la más sencilla de cuantos se encuentran en la aplicación, ya que consta de tan solo la siguiente sentencia: ***time.Stop();***. La palabra time es una instancia de un contador, que se usa para controlar un evento que se explicará más adelante. *Stop( )* es un método de la clase *Timer* (a la que pertenece el objeto *time* antes citado). Mediante este método se cambia la propiedad *Enabled* de la clase *Timer* a modo false, desactivando el contador.

El siguiente método es algo distinto a los anteriores, ya que no trata con ningún botón o pulsador. El constructor es el siguiente: ***private void listBox1_SelectedIndexChanged(object sender, EventArgs e).*** Como se puede deducir del nombre, el evento se activa al seleccionar un elemento de un control de tipo lista o, para ser más exactos, cuando se selecciona un elemento distinto al actual. La rutina consta de una sola sentencia que lee el texto seleccionado en la lista y lo reescribe en el cuadro de texto de la parte inferior de la ventana. La sentencia en cuestión es la que sigue: ***textBox1.Text = listBox1.SelectedItem.ToString();*** . Se asigna el valor de SelectedItem (el elemento seleccionado) convertido a tipo string a la propiedad *Text* del objeto *textbox1*. La propiedad *Text* es la que establece el texto representado en el cuadro.

El cuarto evento es el asociado al botón de Captura, por lo que es muy similar a los anteriores. Este evento es la pieza clave de la obtención de los datos. La rutina comienza activando el contador con la sentencia ***time.Enabled = true;*** que ya se ha comentado anteriormente. Una vez activado el contador, es necesario establecer un tiempo entre iteraciones. Esto se consigue asignando el valor deseado a la propiedad *Interval* de la siguiente manera: ***time.Interval = intervalo * 1000;*** (donde intervalo es el tiempo entre iteraciones). Es importante tener en cuenta que esta propiedad solo acepta valores en

milisegundos. En este caso se asigna al intervalo el valor de 1 segundo para que la primera iteración sea rápida, ya que si no se hiciera de esta manera se tardarían 10 minutos (o el valor que se le quiera dar al intervalo) en poder visualizar por primera vez los datos obtenidos. Tras esto, se asocia el evento *Tick* del contador a una rutina. Para terminar, se comprueba que el archivo que va a ser escrito con los nuevos datos no exista, y si existe se incrementa en 1 el índice del archivo, con lo que no se reescribe ningún archivo. Gracias a esto, aunque se cierre el programa, la numeración de los archivos seguirá el orden normal. Aunque se explicará con mayor profundidad más adelante, para facilitar la comprensión de esta acción es importante conocer que el formato de los nombres de los archivos será el siguiente: Día-mes-año índice.xml. Por lo que el segundo archivo capturado el día 3 de mayo de 2010 tendrá el nombre: 3-5-2010 2.xml.

Como se ha visto, en el método anterior se configuraba un contador de tiempo. El siguiente evento es el asociado a lo que se conoce como *Tick* del contador, que se activa cuando el contador cambia de valor. La primera acción que se lleva a cabo es comprobar que el cuadro de texto en el que se pide el intervalo ha sido rellenado. Si el cuadro está vacío se le da el valor por defecto, que se ha predefinido con 10 minutos, si no, se lee el valor y se inserta en la variable *intervalo* que a su vez se asocia a la propiedad *Time.Interval*. Para insertar el valor del cuadro de texto, que tiene formato de texto, en la variable intervalo, que tiene formato de número entero, se utiliza la sentencia *valorentero = Convert.ToInt32(valortexto)*, donde *valorentero* es el valor resultante en formato de número entero y *valortexto* es el valor inicial, que puede ser de formato texto u otro de los soportados por esta clase, entre los que se incluyen el formato Char (caracteres Unicode), formato DateTime (fechas y horas), formato Double (coma flotante precisión doble) y muchos otros.

4.4.2.1 Conexión y descarga

El siguiente paso es realizar la conexión al módulo. La obtención de los datos puede hacerse de dos maneras distintas, bien mediante FTP o bien mediante protocolo HTTP (ambos de nivel aplicación en modelo OSI). Usar FTP permite descargar un archivo llamado *$onlasc.dat* que contiene únicamente los valores puntuales obtenidos de los sensores. Se podría descargar este archivo y mostrarlo sin ningún tipo de procesamiento con un resultado aceptable, sin embargo la conexión FTP con estos módulos requiere de autentificación con un nombre de usuario y contraseña, lo que la ralentiza. Para resolver este problema se dispone de la opción de descarga con HTTP. De esta manera, es posible descargar el archivo *value.htm* que contiene los mismos valores que *$onlasc.dat* pero con un formato de pagina web, lo que obligará a discriminar las partes irrelevantes (las que no contengan datos) antes de mostrarlo. Utilizando HTTP el programa será un poco más complejo pero mucho más rápido. Si, como en este caso, lo que se busca es presentar los valores en tiempo real, utilizar este tipo de conexión es la opción más acertada de cuantas se ofrecen. Para llevar a cabo la conexión se utiliza la clase *WebClient*. En primer lugar se crea una nueva instancia de la clase de la forma habitual: ***WebClient myWebClient = new WebClient();*** tras esto, se realiza la descarga mediante el método *DownloadFile*, como se ve en la siguiente sentencia: ***myWebClient.DownloadFile(myStringWebResource, fileName);***. Obviamente, antes han debido crearse las variables ***myStringWebResource*** y ***fileName***. La variable

filename es de tipo string y contiene el nombre y la extensión del archivo que se quiere descargar (por ejemplo *value.htm*). La variable *myStringWebResource* también es de tipo string y contiene la dirección web a la que se debe acceder para descargar el archivo y el nombre del archivo y su extensión. La dirección web puede escribirse de dos maneras, usando el nombre de dominio (por ejemplo http://www.google.es/), o como en este caso utilizando la dirección IP del módulo (http://192.168.2.18/). Si se requiere que el módulo sea accesible desde Internet, deberá estar asociado a una IP pública, que se deberá comprar a un proveedor de servicios de Internet. En caso de que solo vaya a ser accedido desde una red privada o local bastará con acceder a su IP privada. Una vez hecho esto, la descarga se realizará y se podrá proseguir con el proceso.

4.4.2.2 Procesado de los valores

Es el momento de discriminar los datos de las cabeceras del archivo y de guardar los valores de una forma más estructurada. En primer lugar se abre el archivo y se lee línea por línea con ayuda de la sentencia siguiente: ***StreamReader sr = new StreamReader("value.htm")***. La clase *StreamSeader* es capaz por si sola de abrir el archivo y leer los caracteres secuencialmente basándose en una clase abstracta llamada *TextReader*. Para dividir el texto en líneas se utiliza la sentencia *ReadLine*. Cada línea se almacena en la variable de tipo string *line*, se le añade un marcador de final de línea (para que más adelante sea más sencillo separarlas de nuevo) y se almacena en otra variable tipo string. Al tener el texto almacenado en una variable, es posible evitar el tener que leer y escribir en un archivo cada vez que se haga algún pequeño cambio.

En este punto se tiene todo el texto del archivo almacenado en una variable separado por líneas. Antes de seleccionar los datos, se debe crear el archivo donde serán almacenados (en este caso si conviene escribirlos en un archivo, ya que esta será su ubicación final), para ello utilizamos la clase *XmlWriter*, que crea un archivo con extensión *.xml* y escribe en él. Este tipo de archivos permiten almacenar datos asociados a etiquetas, lo que hace que permite una organización y estructuración de los datos que sería imposible conseguir con otros tipos de archivos. Además, pueden servir como pasarela a aplicaciones de creación de bases de datos, por lo que si fuera necesaria una ampliación en este sentido esta se vería simplificada. Como ya se ha comentado anteriormente, el archivo lleva la fecha y el número de captura en su nombre, de esta manera será fácil de identificar si hay que acceder a él manualmente. Para crear él nombre utilizamos una serie de sentencias de este tipo: ***DateTime.Now.Day.ToString()+"-"*** . Como se puede observar, se captura el día actual, se convierte en texto y se le añade un guión de separación. Después se capturará de la misma manera el mes y el año y para terminar se le añade un dígito que corresponderá con él número de captura. Para conseguir este dígito, simplemente se comprueba si ya existe algún archivo con la misma fecha, en caso de que así sea se le asigna el número siguiente y se vuelve a comprobar si coincide con el de algún otro archivo.

Los archivos Xml se escriben asociando los datos a etiquetas. Los archivos siempre empiezan por un elemento de inicio, del que cuelgan el resto de etiquetas, este primer elemento se escribe de la siguiente manera: ***writer.WriteStartElement("Captura");*** donde writer es una instancia de la clase XmlWriter y la parte entre comillas será el

nombre de la etiqueta. El primer dato que se escribirá en el archivo será la fecha y la hora de la captura. Aunque la fecha ya consta en el nombre del archivo puede resultar práctico tener la fecha y hora exactas del momento de la captura, sobre todo a la hora de representar los datos en gráficas históricas. Para escribir este elemento se utiliza la siguiente sentencia: ***writer.WriteElementString("tiempo", DateTime.Now.ToString());*** donde la parte entre paréntesis consta de dos elementos, el primero (entre comillas) será el nombre de la etiqueta y el segundo el dato. En este caso el dato es la fecha actual capturada mediante el comando *Now*.

Para escribir los datos en el archivo se toma el texto almacenado en la variable de tipo string, se separa en líneas y se almacena cada línea en su posición correspondiente de un array de datos del tipo string. Estas tres acciones pueden aunarse en una sola sentencia: ***string[] split2 = partido.Split(new Char[] { '/' });***, donde *split2* es el array, *partido* es la variable en la que está almacenado el texto y el carácter entre los apostrofes es el separador de líneas que se había insertado con anterioridad. Discriminar las líneas que contienen dato de las que no lo contienen es muy sencillo, ya que si abrimos el archivo original con un procesador de texto observamos que las líneas que contienen datos tienen la siguiente estructura: *8) Analog_Input_5 10.000 V*. En cada línea encontramos un índice que muestra el número de entrada comenzando desde el número uno. El segundo elemento es el nombre de la entrada, el tercero es el valor y el cuarto son las unidades. El elemento común de todas las líneas de este tipo es el índice seguido de un paréntesis, por lo que se irá separando cada línea en palabras y comparando la primera de ellas con una variable que contendrá un contador (que aumentará su valor cuando se dé una coincidencia) y un símbolo de paréntesis. La comparación se lleva a cabo mediante una sentencia condicional del tipo *if*. Cada vez que se encuentre una línea con dato, almacenaremos la tercera palabra de la línea en una posición de un array de datos tipo string, ya que sabemos por el formato del archivo que los valores capturados siempre se encuentran en dicha posición.

La escritura de los datos se efectúa de la siguiente manera: ***writer.WriteElementString("entrada" + j3, split4[2]);***, donde *j3* es el contador que sustituye al índice original y *split4[2]* es el dato, al que previamente se le han adjuntado unidades simplemente eliminando el espacio entre los dos. El resultado de la escritura de una línea en el archivo es el siguiente: `<entrada10>24.137ºC</entrada10>`. Se escriben secuencialmente todas las entradas. Para finalizar la escritura del archivo, primero es necesario escribir un elemento de finalización, que en este caso marcará el final de la etiqueta Captura (de la que colgaban todas las entradas). La sentencia que realiza esta operación es *WriteEndElement( )*. Los procesos de escritura y lectura de archivos no se cierran automáticamente, sino que siguen ejecutándose en espera de nuevos datos. Para cerrar estos procesos se utiliza habitualmente el método *Close()*, que como se ha podido comprobar está presente en muchas de las clases de C#. En este caso, no se utiliza dicho método ya que la instancia a la clase *XmlWriter* fue declarada utilizando una instrucción *using*. Esta instrucción crea una subrutina, en la que pueden utilizarse objetos y ejecutarse métodos de manera ordinaria, sin embargo, al llegar al final de la subrutina todos los objetos que estén haciendo uso de recursos deberán liberarlos, esto obviamente, significa que los procesos se detendrán. Es por esta razón que no es necesario utilizar en este caso el método *Close()*, aunque podría haberse hecho sin problema alguno. La sentencia *Flush()* que se puede observar antes de la

finalización de la instrucción *using* deja en espera al objeto de escritura. Esta sentencia se utiliza en esa posición para evitar fallos al terminar la escritura, sin embargo el uso habitual de este método es el de jerarquizar los datos escritos en varios grupos. En este caso podría utilizarse para insertar varias capturas en un solo archivo, en el que constarían las etiquetas Captura1, Captura2, etc., y de ellas colgarían los datos recogidos en cada una. Este sistema tendría como contrapartida que en caso de que ocurriera algún fallo en la escritura del archivo, la cantidad de datos que se perderían podría ser muy alta.

Para mostrar los datos de una forma que sea comprensible, además de los propios valores es necesario mostrar los nombres de las entradas asociados a ellos. Para ello, se realizan las mismas operaciones que para guardar los datos, por lo que no será necesario entrar en demasiados detalles sobre el proceso. Sin embargo, sí que existen unas pequeñas diferencias que merecen ser comentadas. Por ejemplo, el nombre del archivo no irá cambiando, sino que será fijo y se reescribirá cada vez que lleguen nuevos datos. Esta forma de proceder es práctica y viable ya que los nombres de las entradas no cambiarán muy a menudo y lo que realmente resulta interesante es tener los nombres actuales, por lo tanto con tener un solo archivo será suficiente. La otra modificación digna de mención es la referente a la posición de la línea en la que están situados los nombres de las entradas. Como se ha observado anteriormente, los valores se encontraban en la tercera posición, sin embargo los nombres estaban siempre en segunda posición, si antes se recuperaban los datos con la palabra *split4[2]*, en esta ocasión se hará con *split4[1]*. El índice está desplazado debido a que este tipo de variables comienzan en 0. Es importante también resaltar que los nombres de las entradas deben consistir de una sola palabra o en su defecto, en vez de separarlas mediante un espacio deberán separarse mediante una barra baja (carácter '_'), ya que de no hacerlo podrían producirse errores en el nombre. Probablemente el nombre aparecería sin los espacios con que se ha escrito, haciéndolo poco inteligible.

4.4.2.3 Representación

La última parte del evento de Tick es la de la representación de los datos. Ya se han descargado y almacenado todos los datos y nombres en sus respectivos archivos y es el momento de presentar los datos para que sean visualizados. En primer lugar, se borra cualquier dato que pueda haber en el cuadro de texto en el que se hará la representación, ya que de no hacerlo los nuevos datos se irían añadiendo al final de la lista y la visualización sería confusa. En realidad, el cuadro de texto es un control de tipo lista, por lo que cada entrada estará asociada a una posición. Además, las entradas que se representarán dependen del módulo que se haya elegido en la casilla designada a tal efecto. La información sobre a que módulo corresponde cada entrada está almacenada, como se ha visto antes en el apartado dedicado a la ventana de configuración en el archivo *entconfig.xml*.

La recuperación de datos guardados en un archivo de formato *Xml* es distinta al de los archivos de texto, ya que como se ha visto estos últimos no se organizan a partir de etiquetas. Aunque el sistema es algo más complejo de lo normal, facilita mucho la recuperación de datos concretos, lo que hace que su utilización sea preferible. El proceso de lectura de estos datos consta de cuatro sentencias. La primera crea una nueva

instancia de la clase *XmlDocument*, que se llamará *xmldoc3*, de la forma habitual. La segunda carga el archivo entconfig.xml mediante el método Load. Este método tiene una peculiaridad con respecto a otros similares, y es que puede cargar archivos que no estén almacenados en el disco duro local, sino en un directorio externo como podría ser una página web. Para hacerlo, simplemente se situaría la URL(dirección web) asociada a la página web en cuestión de la siguiente manera: ***xmldoc3.Load("http://www.google.com/");***. Esta compatibilidad con Internet es comprensible ya que este lenguaje, o metalenguaje más bien, fue diseñado con un enfoque muy dirigido a esta aplicación y precisamente es donde su uso está más arraigado.

La tercera sentencia del proceso de recuperación de los datos crea una nueva variable del tipo *XmlNodeList* y le asigna un valor recogido mediante un método llamado *GetElementsByTagName*, como se puede observar en la sentencia que sigue: ***XmlNodeList config = xmldoc3.GetElementsByTagName("e" + k + "_Modulo");***. Donde *config* es la variable, *xmldoc3* es la instancia de la clase *XmlDocument* y la parte entre paréntesis es el nombre de la etiqueta de la que extraeremos el valor. En esta sentencia se introduce el índice de la entrada de la que se quiere conocer el módulo al que pertenece mediante la variable k. Al hacerlo se busca la etiqueta e15_Módulo (por ejemplo para la entrada número15) en el archivo. La última sentencia simplemente asocia el número recogido a una variable tipo *string* mediante la propiedad *InnerText*. Se realiza esta operación para cada entrada.

Cuando se conoce el módulo al que pertenece, se compara el índice del módulo con el que se quiere representar, que constará en el cuadro de texto rotulado como Módulo. Si coinciden querrá decir que la entrada pertenece al módulo que se desea representar, y por lo tanto habrá que mostrarla. Se utiliza el mismo proceso que el explicado en los párrafos anteriores para abrir el archivo de nombres Noment.xml y se extrae el nombre de la entrada. De nuevo se aplica el mismo proceso para extraer el valor de la entrada del archivo de datos que se acaba de capturar. Una vez se tienen el nombre y el valor se unen en una sola cadena de la forma: *NombreEntrada: 10V*. La separación entre el nombre de la entrada y el dato es un espacio de tabulación y se han añadido los dos puntos por el aspecto visual que otorgan.

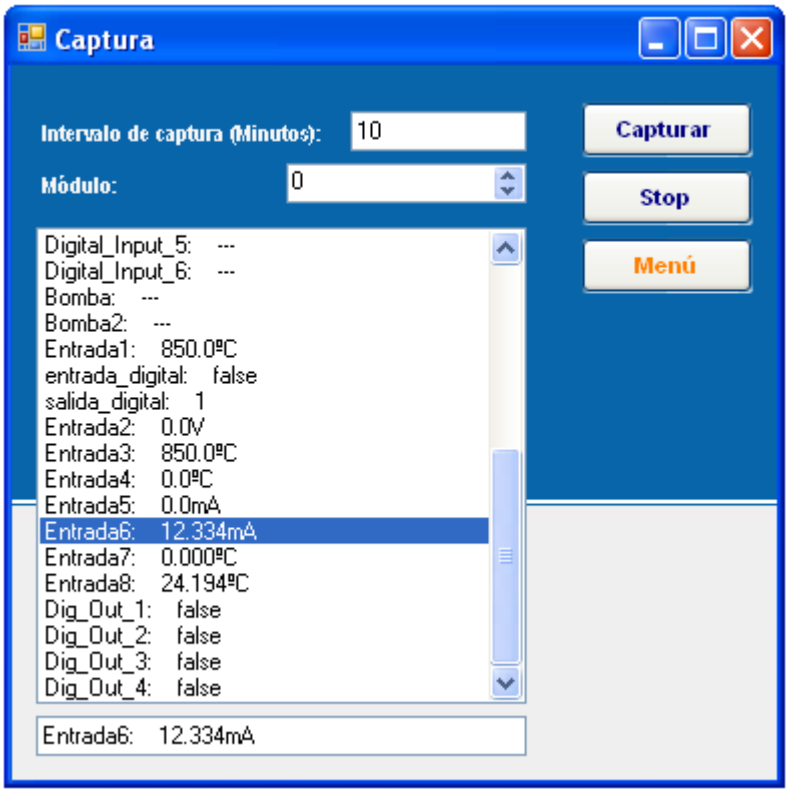
Como se ha comentado antes, el cuadro de texto es un control de tipo lista, por lo que para añadir los strings necesitamos un método específico de la clase *ListBox*, que es la que controla este tipo de objetos. Existen varias maneras de llevar a cabo la adición de elementos a un control de tipo lista, y todas ellas derivan de la propiedad *Items* de la clase citada. Esta propiedad se comporta como una clase en si misma y tiene métodos derivados de ella, todos enfocados al control y modificación de los elementos contenidos en el objeto del tipo *ListBox*. Entre las acciones que se pueden llevar a cabo se encuentran la de borrar todos los elementos, comparar dos elementos entre si, borrar un elementos de la posición especificada y tres métodos distintos para agregar elementos a la lista. El primero es el método Insert, que permite añadir un elemento a una posición determinada de la lista. Es posible utilizarlo, pero tener que asignar una posición fija a cada elemento lo hace poco práctico para esta aplicación. El segundo es el método AddRange, que permite añadir un grupo de elementos a la lista. Puede parecer una buena opción, sin embargo este método obligaría a tener todas las líneas

que se quieran añadir definidas antes de empezar con la adición y por la estrategia que se ha seguido a la hora de crear los strings (uno por uno mediante un bucle for, como se especifica en el párrafo anterior) esto requeriría la creación de una nueva variable y añadir complejidad a la estructura. El último de los métodos es el método Add, que es el que se ha seleccionado para esta aplicación. El método Add agrega un elemento especificado en la primera posición libre de la lista, por lo que va creando la lista añadiendo los elementos al final. La compatibilidad con el bucle for anterior es total, ya que a medida que se van creando los nuevos strings se van añadiendo. Esta es la manera en que se ha implementado en este caso, sin embargo existe otra posibilidad con este mismo método y es usando las sentencias BeginUpdate y EndUpdate, que pretenden evitar que la lista se muestre incompleta retrasando la adición de los elementos hasta que no hayan sido creados todos. Esto no es necesario en nuestro caso, ya que la cantidad de elementos en la lista no será muy amplia y por lo tanto el intervalo de tiempo entre la aparición del primer elemento y la del último no es apreciable. Además, la representación por módulos reduce aún más la cifra de de entradas que representar en cada toma de la lista. Si en un futuro el número de entradas crece mucho (probablemente para que el efecto fuera incómodo el número de entradas debería ser del orden de 500 o más) esta variación podría aplicarse al programa para mejorar la experiencia del usuario.

Para terminar con la representación se ha añadido la posibilidad de mostrar las entradas de todos los módulos a la vez. Esta funcionalidad se activa al no seleccionar ningún módulo para la representación o al seleccionar el módulo número cero. De esta manera evitamos posibles errores por falta de atención. La representación se lleva a cabo de forma análoga a la anteriormente explicada exceptuando la comparación con el número de módulo, es decir, si el módulo seleccionado es cero se muestran todas las entradas. Existen cuatro entradas, que se sitúan al comienzo de la lista, que no pertenecen a ningún módulo en concreto ya que hacen referencia a contadores abstractos. Se ha decidido representarlas en esta vista por completitud y debido a que pueden servir como referencia de funcionamiento cuando los valores están estáticos, aunque no aportan mayor información.

4.4.3 Aspecto final

El resultado final de la representación de los valores es el siguiente:



4.5 Gráficos

4.5.1 Descripción

Es la clase que representa la ventana de Gráficos. En ella es posible visualizar los valores captados en nuestra instalación tomando como base el día en que se capturaron. En la parte superior izquierda de la pantalla se encuentra el selector de entrada. Con él se elige la entrada que se desea visualizar. Debajo de este se pueden observar dos selectores con formato de fecha rotulados *Desde* (el situado en la parte izquierda) y *Hasta* (el situado a la derecha). Estos selectores, permiten elegir el intervalo de días en el que se acotará la visualización. Si se selecciona un intervalo entre el 5 de Marzo y el 7 de Marzo, se podrán visualizar los valores capturados los días 5, 6 y 7. Si se desea visualizar los valores de un solo día simplemente se elegirá el mismo valor para los dos selectores. En la parte central de la ventana se encuentra la gráfica, que es el lugar en el que se representarán los valores. Tiene formato de gráfica lineal con dos ejes. En el eje horizontal se representará el tiempo y en el eje vertical la magnitud de la entrada a representar. Dependiendo de la entrada elegida, el título de la gráfica cambiará para mostrar el nombre de dicha entrada y la rotulación del eje vertical también lo hará para corresponder a la magnitud de la misma. En la parte superior derecha se encuentran tres botones rotulados *Crear Gráfica*, *Volver a menú* y *Ayuda*. *Crear gráfica* ha de pulsarse cuando se han seleccionado las fechas de inicio y final del intervalo a representar, al hacerlo el control de selección de entrada se activará y se podrán ver las entradas disponibles en el intervalo indicado. *Volver a menú* devuelve al usuario al menú principal. Por último, Ayuda despliega una ventana explicativa de cómo configurar correctamente la visualización y las funcionalidades de la ventana.

4.5.2 Diseño

Esta ventana puede dividirse en dos partes, una primera que contiene las funciones habituales de un formulario Windows típico y una segunda que añade las funcionalidades de la representación gráfica. Esta segunda parte está implementada en forma de un método privado, y por lo tanto solo accesible desde esta ventana. A fin de hacer que el funcionamiento de esta ventana sea más sencillo de comprender, es conveniente retrasar la explicación sobre el método de la representación hasta haber comprendido el resto de secciones de la ventana. Por lo tanto, se comenzará explicando los eventos asociados a los botones y demás objetos comunes.

4.5.2.1 Formulario Windows

4.5.2.1.1 Botón Volver a Menú

Una vez creada la clase e inicializado el Form5, lo primero que se encuentra es el evento asociado al clic en el botón *Volver a Menú*. Como el resto de botones rotulados de esta manera que se encuentran en esta aplicación, su función es la de cerrar la ventana actual y abrir la ventana de menú principal. Estas funciones se llevan a cabo apoyándose en la propiedad *Visible* y el método *Close* de la clase del formulario actual.

4.5.2.1.2 Botón Ayuda

El siguiente es el evento del botón Ayuda. Este evento abre un nuevo Form, en el que se puede leer una pequeña guía de uso de la aplicación. Seleccionando la pestaña correspondiente a la representación gráfica se puede leer la parte de esta guía que trata el contenido de esta ventana.

4.5.2.1.3 Botón Crear Gráfica

El siguiente evento es el que controla el clic del botón Crear Gráfica, que es donde comienza el proceso de la representación gráfica. En primer lugar se el contenido de el control tipo lista desplegable rotulado con el nombre Entrada. De no hacerlo, al cambiar de fechas de fechas para la representación, las nuevas entradas se situarían después de las antiguas resultando en una lista con los nombres de las entradas repitiéndose el número de veces que se haya cambiado de fechas. Como norma general, esta acción deberá llevarse a cabo siempre que se trate de cargar nombres u otros datos en un control. Lo siguiente es crear un bucle *for* que termine cuando el número de iteraciones iguale a la cantidad de entradas conectadas. En cada repetición del bucle cargaremos el nombre de una de las entradas y lo añadiremos a la lista. Para llevar a cabo este proceso es necesario en primer lugar crear una nueva instancia de la clase XmlDocument y con ella cargar el archivo que contiene la información del nombre de las entradas (en este caso el archivo Noment.xml). Con ayuda del contador del bucle *for*, para cada entrada se busca la etiqueta que corresponda y se almacena el nombre en la variable Nombres, que se ha creado para este fin. Acto seguido se recupera el dato que se acaba de almacenar y se le asigna a una variable del tipo string mediante la propiedad InnerText, como se ha explicado en apartados anteriores de este mismo documento. Una vez hecho esto, se introduce el dato en el control desplegable mediante el método Add, de la misma manera en que se hacía en el formulario de captura y representación en tiempo real de los datos. Este proceso da como resultado un control que contendrá todos los nombres de las entradas existentes, que podremos seleccionar haciendo clic sobre ellos.

Este tipo de controles, conocidos en C# como ComboBox, son muy habituales en formularios y aplicaciones de tipo Windows, ya que permiten visualizar todas las opciones existentes a la hora de seleccionar una de ellas. Además de esto concentra en un solo elemento las capacidades de un cuadro de texto y de un control de tipo lista común pero en menor espacio, ya que al desplegarlo se hace visible la lista de objetos contenidos en el control, pero también es posible escribir en el ComboBox y este buscará la opción más cercana a la cadena proporcionada dentro de la lista. Este potencial hace de este tipo de control uno de los más interesantes, razón por la cual se ha utilizado en diversos puntos de la aplicación. Sin embargo, se ha intentado en todo momento que el rango de recursos utilizado para crear el programa fuera los más amplio posible, ya que de esta manera el aprendizaje que ha supuesto el desarrollo de la aplicación ha sido mayor.

4.5.2.1.4 Selección de fechas

El último evento de este formulario se produce al seleccionar una de las entradas del ComboBox. En él se capturan las fechas de inicio y final que previamente habrán sido definidas por el usuario en sus respectivos cajetines (rotulados Desde y Hasta) y da acceso al método privado de la representación gráfica.

En este evento se opera con los selectores de fecha. Estos selectores tienen la forma de un ComboBox con el formato de días, meses y años, que nos servirán para fijar los límites de la representación.

Existen varias maneras de proceder, por ejemplo en un primer momento se pensó en crear una pequeña pantalla para la selección de las fechas con un calendario fijo (no desplegable). Una de las particularidades de este sistema era la posibilidad de seleccionar dos fechas en un solo calendario ya que contar con dos calendarios desplegados continuamente ocuparía demasiado espacio visual. Este sistema se llevó a la práctica y aunque los resultados fueron buenos, la selección no era muy intuitiva y la complicación de abrir una ventana simplemente para seleccionar las fechas lo hacía poco práctico. Una solución que se barajó fue la de incorporar la adquisición de los datos en esta ventana, sin embargo esto hubiera significado desestructurar la aplicación y hacerla más compleja tanto para los usuarios como para programadores que pudieran analizar el código para llevar a cabo mejoras o actualizaciones. Por lo tanto se optó por mantener el orden lógico del programa y se buscó la opción de integrar los calendarios de manera reducida en la propia ventana de representación, consiguiendo que fueran más accesibles y el proceso para visualizar los históricos, por parte del usuario, se vio simplificado.

En primer lugar, este evento captura las fechas de inicio y final del intervalo de representación que han sido definidas por el usuario. La clase a la que pertenecen los selectores de fecha es DateTimePicker, que contiene una propiedad llamada Value. Esta propiedad permite visualizar la fecha elegida en formato DateTime y se implementa de la siguiente manera: ***fechaini = dateTimePicker1.Value;***. Se repite la misma operación para la fecha final del intervalo y se comparan. Comparar las dos fechas es necesario para determinar el número de días que se deben representar en la gráfica, y obviamente existen varias maneras de determinar este número.

La primera opción que se barajó fue extraer los días elegidos, convertirlos a formato de número entero y restar el número de día final y el inicial. Esta opción se descartó de inmediato, ya que no permitiría representar un intervalo de fechas de meses distintos, ya que en la resta solo tomaban parte los días y por lo tanto, no habría posibilidad de discriminar el mes al que pertenecerían.

También se barajó la posibilidad de numerar los días del año, desde el 1 de enero hasta el 31 de diciembre asignando a cada día un número. Esto permitiría mostrar las fechas de varios meses, pero no de varios años. Puede parecer que visualizar los datos de varios años no es algo habitual, sin embargo es posible que se requiera esta operación, además podrían generarse problemas similares al de la opción anterior si se piden los datos de fechas como por ejemplo: el 1 de febrero de 2010 y el 2 de febrero de 2011. En

este caso el programa mostraría tan solo dos días, cuando en realidad se requería todo un año.

A medida que el proceso de aprendizaje avanzaba, se observó que existía una manera más adecuada de llevar a cabo la operación requerida, que había de hacerse en dos fases como se explicará más adelante. Para ello, es necesario obtener en primer lugar un dígito que indique si la fecha final del intervalo es posterior o igual a la de inicio (también conviene saber si es anterior, pero no tendría sentido que lo fuera). Tras estudiar algo más en el tipo de datos *DateTime* se dio con la solución que finalmente se implemento. Existen dos métodos asociados a este tipo de datos llamado *Compare To* y *Compare*, que se observó podrían llevar a cabo la operación requerida. En la práctica, ambos métodos cumplen el mismo cometido, con la diferencia de que *Compare To* permite comparar elementos de tipo objeto y también de tipo *DateTime*, mientras que *Compare* solo permite comparar elementos de tipo fecha. Por lo tanto se podría decir que *Compare* es una simplificación para un caso concreto de *Compare To*.

Ambos métodos devuelven un número entero que indica que fecha de las comparadas es anterior a la otra o si son iguales. Es importante que las fechas sean comparadas en el orden correcto, ya que de lo contrario el número resultante tendrá el signo invertido, como se puede ver a continuación: ***numcapturas = fechafin.CompareTo(fechaini);***. Numcapturas es la variable de tipo entero que recogerá el valor resultante de la comparación, fechafin es la fecha final en formato DateTime y fechaini la fecha inicial en el mismo formato. Si se implementa de esta manera, el resultado de la comparación siempre será positivo(+1), ya que la fecha final siempre debería ser mayor que la fecha final, o como caso extremo podrían ser iguales con lo que el resultado sería cero, pero nunca negativo(-1). Este sistema, unido a una segunda parte que se estudiará junto con el método privado de representación, lleva a cabo adecuadamente la tarea que se pretendía realizar de una manera sencilla.

4.5.2.1.5 Máximo y mínimo

Antes de finalizar el método con la llamada a la representación, se decidió añadir la lectura de los valores máximo y mínimo (para alarmas) contenidos en el archivo entconfig.xml. Una vez recuperados, estos valores se escribirán en los cuadros de texto rotulados como “Val. Máximo” y “Val. Mínimo” que se encuentran en la parte inferior de la ventana, debajo de la gráfica. Como ya se explicó, estos valores servirán para facilitar la futura inclusión de alarmas en la aplicación. Por ahora, simplemente se utilizarán como referencia para que el usuario sea capaz de interpretar los valores a primera vista. La extracción de los valores se hace de la manera típica para un archivo xml, como se puede ver a continuación:

```
XmlDocument xmldoc3max = new XmlDocument();
xmldoc3max.Load("entconfig.xml");
XmlNodeList maximo = xmldoc3max.GetElementsByTagName("e" + comboBox1.SelectedIndex + "_Max");
string maxi = maximo[0].InnerText;
textBox1.Text = maxi;
```

En la primera y segunda línea se crea la instancia a la clase del lector de archivos Xml y se abre el archivo en sí. En la tercera línea se extrae el valor correspondiente a la etiqueta “e”+número de entrada+”_Max” (que daría por ejemplo: “e5_Max”) y se introduce en una variable de tipo XmlNodeList. En la siguiente sentencia se convierte el valor a tipo string mediante una conversión implícita. La última línea es la que se encarga de insertar el valor en el cuadro de texto correspondiente. Esta última sentencia es importante ya que a la hora de incorporar el valor en la gráfica será de este cuadro de texto de donde se recogerá el valor. Además, se ha previsto que si el máximo y el mínimo no existen o no tienen el formato correcto se muestren ceros con el fin de no entorpecer al visualización de los resultados.

4.5.2.1.6 Llamada a la representación

Por último, se ha de llamar al método de representación. Al llamarlo, se le enviarán una serie de datos que serán necesarios para el funcionamiento de la rutina. La llamada se hace de la siguiente manera: ***CreateGraph(zedGraphControl1, comboBox1.Text, comboBox1.SelectedIndex,numcapturas);***. CreateGraph es el nombre del método. Los datos que se pasarán al programa son cuatro: En primer lugar zedGraphControl1, que es el nombre del objeto que conforma el cuadro de la gráfica. En segundo lugar se envía el nombre de la entrada que se desea representar, para ello se extrae el string seleccionado en el ComboBox con la sentencia comboBox1.Text. El tercer dato es el índice de la entrada *comboBox1.SelectedIndex*, que como se ha explicado previamente, sirve para localizar la entrada dentro del archivo mediante su etiqueta. Por último se envía el número resultante de la comparación, que es lo que determinará el número de muestras que se han de representar.

4.5.2.2 El control para representación gráfica ZedGraph

En este punto, el programa salta a la rutina de *zedGraphControl1*. Esta rutina es un método privado, es decir, solo accesible desde la clase del formulario actual y del tipo *Void*, lo que significa que no entregará ningún valor al finalizar su ejecución. Este método se define de la siguiente manera: ***private void CreateGraph(ZedGraphControl zgc,string nombreentrada,int indiceentrada,int numeromuestras)***.

Este método realiza la representación y sigue siguiente estructura: En primer lugar se evaluará si la fecha de final es distinta de la fecha inicial y su separación en días. Si es distinta, es decir si la separación es mayor que cero, se ejecutará una sola vez un bucle en el que se llevará a cabo la recuperación y representación de los datos. Al finalizar el bucle se decrementará en una unidad la diferencia en días entre las fechas y se volverá a evaluar las fechas son distintas. En caso positivo, se repetirá la acción del bucle. Si la separación es cero también se ejecutará el bucle, con la diferencia de que al finalizar a la separación se le dará valor negativo. En caso de que la separación entre las fechas sea negativa el bucle no se ejecutará.

Por lo tanto, para representar los valores capturados en el intervalo de fechas seleccionado mediante este sistema, es necesario conocer la diferencia de días entre la fecha inicial y la final. Para ello se habían barajado ciertas posibilidades, expuestas anteriormente, pero ninguna había funcionado correctamente. Al estudiar el tipo de

datos *DateTime* para hacer la comparación también se encontró un método llamado *Subtract*, que resta dos fechas o intervalos de tiempo. El método Subtract devuelve un dato en formato *TimeSpan*, el cual tiene una estructura peculiar como la siguiente: *d.hh:mm:ss*, donde *d* son los días, *hh* las horas, *mm* los minutos y *ss* los segundos, todos ellos referidos al intervalo de tiempo. De esta manera si realizamos la resta (con el método Subtract) de dos fechas como por ejemplo el 3 de Octubre de 2010 a las 00:00:00 y el 1 de octubre de 2010 a la misma hora, el resultado sería el siguiente: 2.00:00:00. O si se resta el 2 de Agosto a las 03:00:00 y el mismo día a las 02:30:00, el resultado sería 0.00:30:00. Debido al formato de este tipo de datos, es muy sencillo separar cada uno de los parámetros que contiene. En este caso, solo son necesarios los días (cuyo valor será almacenado en la variable contdias) ya que se entiende que será el mínimo intervalo de representación deseable, ya que probablemente si fuera inferior el número de muestras sería demasiado reducido para resultar interesante.

Es importante señalar que en este apartado se utilizan métodos y propiedades provenientes de una clase externa a la biblioteca de C#. Esta clase tiene como nombre ZedGraph, y es la que contiene todas las funcionalidades para crear una gráfica. Esta clase ha sido descargada de la página www.zedgraph.org, en la que se puede encontrar toda la información necesaria sobre su uso y funcionalidad. Para aprender los rudimentos básicos de creación de gráficas es recomendable seguir el tutorial que se puede encontrar en la citada página web. Para un listado completo de las clases, métodos y propiedades asociadas a ZedGraph puede visitarse la página de documentación online que puede encontrarse en la siguiente dirección Web: http://zedgraph.sourceforge.net/documentation/default.html.

Se ha estudiado de forma general el propósito y estructura del método, ahora se verá de forma más detallada como se han implementado las acciones descritas. Tras calcular el intervalo de tiempo de la manera descrita en el párrafo anterior, comenzamos a configurar la ventana para la representación. En primer lugar, creamos una nueva instancia de la clase GraphPane, que pertenece a la biblioteca de representación de ZedGraph. Una vez hecho esto, se borra el contenido de la gráfica para evitar sobrescribirlo, en caso de que haya una curva ya dibujada. Tras esto se configuran los ejes.

Como se ha explicado antes, el eje Y de la gráfica representará los valores y el eje X las fechas y horas correspondientes. Los ejes están configurados por defecto para recibir los datos en formato Double y representarlos sin ninguna modificación, por lo que en el caso del eje Y no habrá problemas y no será necesaria configuración alguna. El eje X sin embargo, es distinto, ya que se requiere representar un valor de tipo DateTime, pero el eje solo soportará datos tipo Double. Para solucionar este problema, habrá que realizar una conversión entre estos dos tipos de datos, pero también se deberá configurar el eje X para que interprete el valor Double como una fecha. La configuración del eje se realiza mediante la siguiente sentencia: ***myPane.XAxis.Type*** **=** ***AxisType*.Date;**. Como se puede ver, accedemos a la propiedad Type, que a su vez pertenece a la propiedad XAxis, desde la que se accede a todos los parámetros y acciones del eje X. Si se quisiera acceder al eje Y simplemente se cambiaría la sentencia sustituyendo YAxis en el lugar donde estaba XAxis. En el lado derecho de la igualdad se puede observar como se escribe la nueva configuración, por la que los datos que

entren en la gráfica y sean asociados a este eje serán interpretados como fechas. Para configurar la rotulación de los ejes se accede a la misma propiedad pero cambiando Type por Title.Text. También cambiaremos el nombre del gráfico para que concuerde con le nombre de la entrada que se esté representando. Por último, para que la visualización sea más clara, ocultaremos la leyenda de la gráfica, ya que si solo se representa una variable no será necesaria.

Para insertar los valores en la gráfica se utiliza una matriz con tantas columnas como el número de ejes de la gráfica y tantas filas como puntos se deseen representar. En la biblioteca de ZedGraph, esta matriz se configura con la clase PointPairList, que viene a significar lista de pares de puntos. En este caso la gráfica consta de dos ejes y el número de puntos será variable. Para preparar la representación se crea una nueva instancia de la matriz con el nombre list y un índice llamado comparador al que se le asigna el valor de numeromuestras, que como se recordará contenía el número entero resultante de la comparación entre las fechas de inicio y final. Además de la PointPairList de los valores, también se crearán otras dos, una para el máximo y otra para el mínimo. Una vez hecho esto, la rutina de representación está lista para comenzar.

En primer lugar se evalúa el valor de la variable *comparador* en el comienzo de un bucle While. Si los valores de las fechas han sido correctamente seleccionados (la fecha de inicio siempre ha de ser anterior o igual a la de finalización), contendrá el valor uno o cero. En caso de que así sea la ejecución continuará dentro del bucle. Si el valor es -1 significará que las fechas son incorrectas y no se mostrará ningún valor en la gráfica. Como en los demás lenguajes de programación, el bucle While se ejecuta siempre que se cumpla una condición, que es evaluada cada vez que se va a iniciar una nueva iteración. En este caso la condición es que el valor de comparador sea distinto de menos uno. Si en el curso del programa esta variable tomara dicho valor el bucle se ejecutaría normalmente hasta el final, pero no se iniciaría de nuevo.

Una vez dentro del primer bucle, se encuentra otro bucle también de tipo While. La condición de este es que el nombre de archivo que se va a abrir exista. Como se sabe, el nombre del archivo de datos consta de la fecha y un contador, por lo que utilizaremos este último para abrir todos los archivos del día correspondiente incrementando su valor en cada iteración. La fecha del archivo coincidirá con la fecha de inicio del intervalo de representación. Ya se puede vislumbrar la estructura que sigue el programa. Una vez insertado el nombre del archivo, el segundo bucle va abriendo y representando los datos de cada archivo correspondiente a la fecha del nombre. Al llegar a la última muestra del día el contador vuelve a incrementar su valor, como cada vez, pero al comprobar si el archivo existe la respuesta es negativa, ya que la última muestra es la que contiene el índice más elevado, por lo que no se ejecuta de nuevo. Tras esto, volvemos al final del primer bucle, donde la fecha de inicio se incrementa en un día utilizando el método AddDays y donde comprobamos que el valor de la variable contdias (que contenía el valor del intervalo de días a representar) es distinto de cero. Si es cero significará que solo era necesario representar un día, por lo que no deberá volver a ejecutarse el bucle, lo que se conseguirá dando el valor de menos uno a la variable comparador. Si el valor es mayor que cero, mostrará el número de iteraciones restantes. Se reduce el valor de contdias en una unidad, debido a que uno de los días del intervalo ya ha sido mostrado, pero como el valor era positivo el nuevo valor lo seguirá siendo o tendrá valor cero, lo

que hará que el bucle vuelva a ejecutarse. Al haber incrementado el valor de la fecha inicial (que también se usa para dar nombre al archivo del que provienen los datos en cada iteración), se recuperarán por el mismo proceso los datos del día siguiente.

Aunque la estructura de la representación está detalladamente explicada, falta aún por especificar como se lleva a cabo la escritura en la gráfica, lo que obliga a volver al interior del segundo bucle. Se sabe que la gráfica espera que se introduzcan dos coordenadas por punto, donde la coordenada X será el tiempo y la Y la magnitud. En los archivos de datos se previó esta necesidad y se almaceno junto con ellos la fecha y la hora a la que habían sido capturados los datos del archivo. Por lo tanto, todos los datos necesarios se encuentran en los archivos. Para recuperar estos datos, se realizará el proceso de extracción de datos provenientes de archivos Xml, de la misma manera que en las ocasiones anteriores ya explicadas. En este caso los valores a representar quedarán almacenados en la variable *a* y las fechas correspondientes en la variable *t*. Las fechas ya están adaptadas a la forma en que serán representadas, sin embargo los valores no lo están, ya que fueron guardados junto con las unidades. Por lo tanto, estos datos precisan de un procesamiento.

En algunos casos, cuando la entrada no ha sido correctamente configurada o no está habilitada su valor sigue constando en los archivos de datos, con la diferencia de que el valor que tiene asociado no es numérico, sino tres guiones "---". Estos caracteres han de ser sustituidos ya que el gráfico solo acepta caracteres de tipo numeral. Cuando se encuentra un caso como este, se sustituyen los tres guiones por un cero en la variable a. Sin embargo, esto no da la información de que la entrada no está operativa. Para incluir esto, en vez del nombre de la magnitud, en el eje Y constará N/A (como abreviatura de Ninguna o del ingles Not Available). Un caso similar ocurre con las entradas de tipo booleano, ya que los valores que pueden tomar no son numéricos y no constan las unidades. Una entrada configurada para este tipo de datos puede devolver valor Falso ("False" en ingles) o Verdadero (comúnmente "True"), por lo que para solucionar el problema habrá que asociar ambos valores a un número. En este caso, se han elegido el cero para el valor falso y el uno para el verdadero, pero existen otros programas que utilizan valores negativos para representar los valores falsos. Además de esto, se debe hacer constar que los valores representados son del tipo Booleano. La mejor manera de hacerlo es nombrar el eje Y como Bool.

El siguiente paso en el procesamiento de los datos es reconocer las unidades de la entrada que se va a representar. Como se ha explicado antes, las unidades están unidas a los datos y representadas como norma general por una letra. Así, si la entrada está configurada para recibir voltaje, en las unidades constará una V. Si las unidades vienen representadas por una I, la magnitud serán los amperios (corriente eléctrica). Por último, se encuentra un caso especial, ya que las medidas de resistencia no corresponden a una R o una O (para ohmios), sino que se representan con el símbolo de grados ºC. Esto tiene su razón de ser en el hecho de que los sensores disponibles que entregan un valor de resistencia son del tipo Pt100 o Pt1000, que son los habituales en la medida de temperaturas. Al recuperar los datos del archivo, se buscarán estos caracteres (V, I o C) y dependiendo de cual se encuentre se le asignará una magnitud u otra.

El último paso del procesamiento de los datos es eliminar los caracteres de las unidades. Una vez se ha asignado una magnitud a cada entrada no es necesario que dichos caracteres sigan apareciendo junto a los valores a representar. Por ello, los suprimiremos utilizando el método Replace asociado al tipo de datos string. Este método busca el string especificado dentro de la cadena y lo sustituye por otro string también definido por el usuario. En este caso sustituimos los caracteres que representan las unidades por un string vacío, como en la sentencia siguiente: ***a = a.Replace("V", "");***. Para terminar, se convierten los datos de las variables *a* y *t* a formato Double la primera y DateTime la segunda.

Antes de introducir los datos en la gráfica, se ha de realizar una nueva conversión de formato para la variable que contiene la información de la fecha. Como se ha comentado antes, la gráfica solo acepta datos de formato Double, sin embargo, uno de los datos es de tipo fecha. Realizar una conversión mediante el método Convert entre los formato DateTime y Double, como se hacía en ocasiones anteriores, no es posible. Como se puede apreciar en la biblioteca de Windows MSDN, al tratar de llamar al método *Convert.ToDouble(DateTime)* se produce una excepción del tipo InvalidCastException, que índica que la conversión (explícita en este caso) que se intenta realizar no es válida, huelga decir que tampoco existe conversión implícita posible entre estos dos tipos de datos.

La solución a este problema se encuentra en uno de los métodos de la clase DateTime. El método ToOADate permite convertir datos con formato de fecha tradicional en formato de fecha de automatización OLE. Este tipo de formato de fecha, que forma parte de un protocolo de comunicación entre aplicaciones de Microsoft, tiene la misma forma que el tipo Double, con lo que podrá ser insertado en la gráfica. Para poder almacenar una fecha con formato de coma flotante, el método usa un día y una hora de referencia con respecto al que se calcula cada fecha. Esta fecha es la media noche del 30 de Diciembre de 1899 y el resto del calendario se calcula como la diferencia en días entre dicha fecha y la que se quiera calcular. Por ejemplo, la media noche del 31 de Diciembre de 1899 se representaría como 1,0; las 6 de la mañana del 1 de enero de 1900 se representaría como 2,25 y las 6 de la mañana del 29 de Diciembre de 1899 se convertiría en -1,25. Con este método puede representarse cualquier fecha hasta el último minuto del 31 de Diciembre de 9999.

Una vez realizada esta última conversión los datos pueden añadirse a la lista mediante la sentencia siguiente: ***list.Add(ex, adou);***. Donde *list* es el nombre de la lista que controla los valores a insertar, *ex* es la fecha en formato de automatización OLE y *adou* es el valor recogido del sensor en formato Double. Para insertar los valores de la lista en la gráfica se utiliza de nuevo la biblioteca de clases de ZedGraph. En este caso la clase LineItem, que permite añadir y configurar una línea dentro de la gráfica. La sentencia utilizada es la siguiente: ***LineItem myCurve = myPane.AddCurve("My Curve", list, Color.Blue, SymbolType.Circle);***. El nombre que se le da a la curva de datos es myCurve, el método AddCurve sirve para insertar los datos en la gráfica. Dentro del paréntesis se puede observar que se introduce la variable list, que contenía los datos y también se configuran dos parámetros del aspecto visual de la curva, el color y la forma de representar los puntos de datos.

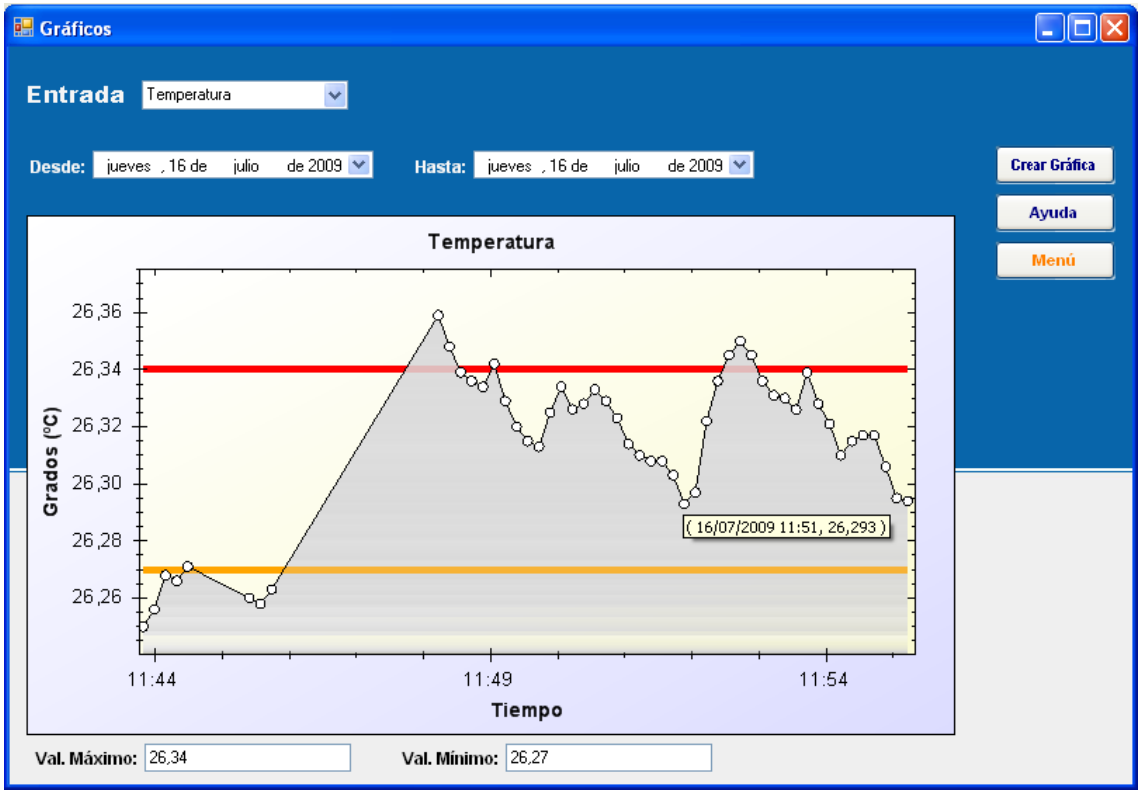
De esta manera podría terminar el proceso de representación, sin embargo, se optó por mejorar algunos detalles estéticos con el fin de que la gráfica sea más atractiva visualmente. Por ejemplo, se pintó el área debajo de la curva de un color con gradiente o se le dio color al fondo de la gráfica, lo que realiza dos funciones, por un lado el aspecto es mejor y por otro, al cambiar el color del fondo cuando se está representando algún parámetro, puede ayudar a comprobar si la gráfica está o no activa. De una manera similar a la utilizada para los valores, se introducen también el máximo y el mínimo a la gráfica y se refina un poco su estética. El código utilizado es el siguiente:

```
listmax.Add(ex, Convert.ToDouble(textBox1.Text));
LineItem myCurvemax = myPane.AddCurve("", listmax, Color.Red, SymbolType.None);
myCurvemax.Line.Width = 5;
```

En la primera sentencia se introducen los valores de tiempo (representados por la variable ex) y el valor máximo, que se encuentra en el cuadro de texto textBox1. La segunda línea inserta la curva creada (en este caso será una línea horizontal en el valor especificado) en la gráfica y le da un aspecto determinado. En este caso será de color rojo y los puntos no estarán marcados. La última sentencia se utiliza para especificar el grosor de la línea. Se ha elegido un grosor de 5 unidades para que sea más visible. Es importante destacar que esta línea se encontrará en un segundo plano dentro de la gráfica, por lo que no obstaculizará la visualización de los datos. Se realiza la misma operación para el mínimo, cambiando su color por el naranja. Como último detalle se hace un escalamiento automático de los ejes para que la visualización sea inmediata.

4.5.3 Aspecto final

El aspecto de la ventana de representación gráfica es el siguiente:



4.6 Ayuda

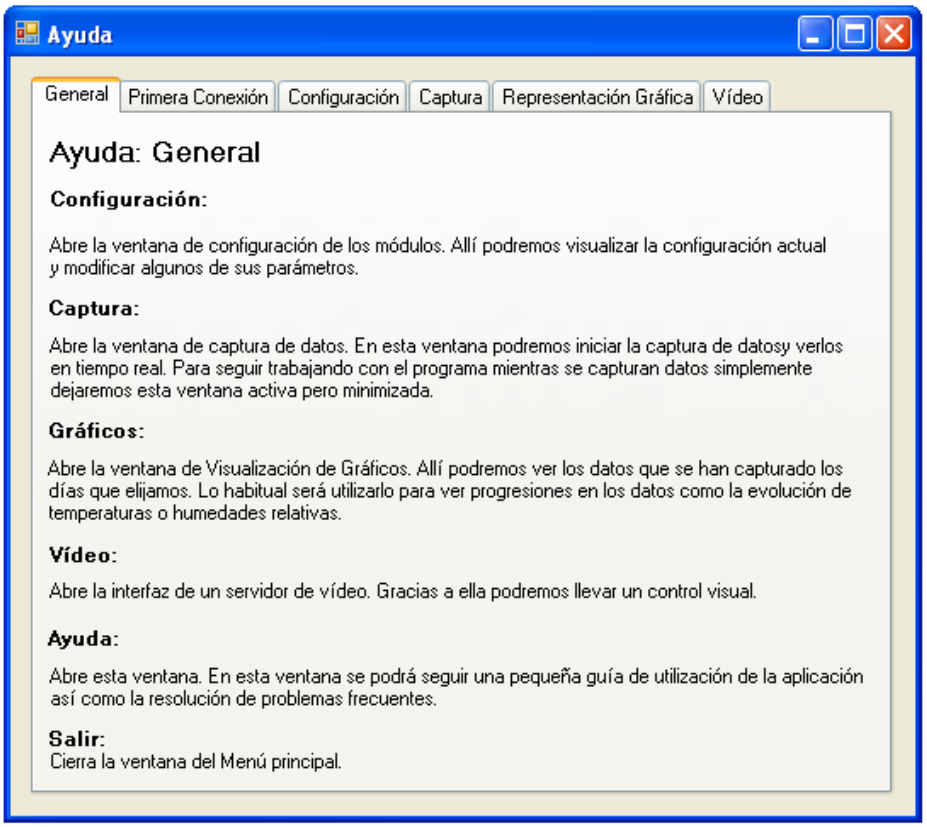
4.6.1 Descripción

Es la clase que representa la ventana de Ayuda de la aplicación. La ventana consta tan solo de un control con cinco pestañas, cada una dedicada a una de las secciones de la aplicación e incluida una para funcionamiento general del programa. Por lo tanto, se tiene una pestaña dedicada a la ventana de configuración, una para la ventana de captura, una para la representación gráfica, otra para la ventana dedicada al vídeo y una última para información general. En estas pestañas se pueden leer explicaciones sobre el funcionamiento de la aplicación así como instrucciones para el correcto manejo de la misma. Se pueden encontrar también algunos ejemplos de cómo realizar las tareas más comunes, como puede ser cambiar las fechas de la representación gráfica.

El acceso a esta ventana puede hacerse desde cualquier punto del programa ya que se ha incluido en cada formulario (salvo en el de bienvenida) un botón que conduce al usuario directamente a esta sección. El cierre del formulario debe hacerse desde los botones incluidos en el marco de la ventana ya que no se ha considerado necesario aumentar el espacio ocupado por la ventana para este fin.

4.4.3 Aspecto final

El aspecto final de la ventana de ayuda es el siguiente:



4.7 Vídeo

4.7.1 Descripción

Es la clase que representa la ventana Vídeo. A través de ella el usuario puede acceder a una interfaz web que le permitirá visualizar vídeo transmitido en directo desde una o varias cámaras web. La ventana no contiene ningún elemento de tipo botón o cuadro de texto. Aunque la visualización de vídeo no estaba contemplada en un principio dentro de las funcionalidades del proyecto, se ha considerado interesante incluirla ya que puede considerarse que la inspección visual de la explotación es un medio de control muy necesario en este tipo de aplicaciones.

4.7.2 Diseño

La ventana cuenta con un único objeto, la aplicación web browser. Esta aplicación, que puede insertarse manualmente en cualquier formulario de tipo Windows forms, permite acceder a páginas web y las muestra en el cuadro correspondiente. En este caso, se ha hecho coincidir dicho cuadro con los bordes del formulario, dando así la imagen de una sola ventana. En una aplicación habitual del web browser, se añadiría un cuadro de texto en el que se permitiera al usuario especificar la página a la que desea acceder. Sin embargo, en este caso se ha decidido que no sea así y automatizar el acceso limitándolo a la página en la que se encuentra la información de la cámara. Esta operación se puede llevar a cabo con cualquier explorador de Internet.

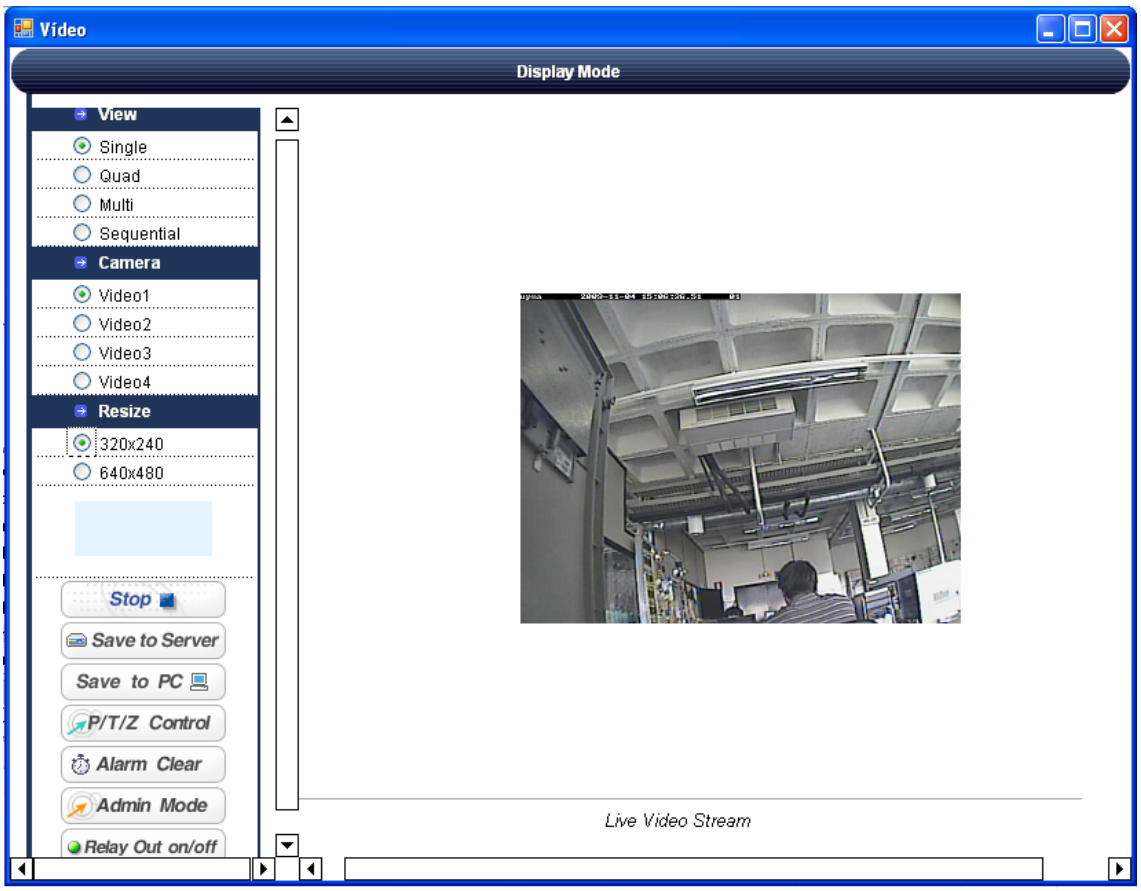
Una vez insertada la dirección IP de la cámara se accede a la página de bienvenida en la que se pide un nombre de usuario y una clave de acceso. Cabe destacar que en esta misma ventana se puede acceder a dos modos de funcionamiento distintos: El modo Administrador y el modo Usuario. El modo administrador se utiliza para modificar la configuración de la cámara, tanto en aspectos de visualización como de acceso a ella. La principal utilidad del modo usuario es la visualización de las imágenes aunque también permite acceder a ciertas opciones de configuración, siendo estas bastante limitadas. La visualización de las imágenes se hace prácticamente en tiempo real, aunque se introduce una pequeña latencia debido a las deficiencias propias del sistema de procesado de la imagen.

El sistema de visualización soporta hasta cuatro cámaras web distintas, que podrán visualizarse una por una o de manera simultánea. La calidad de la imagen puede ser configurada también, pudiendo el usuario elegir entre una resolución de 320x240 o 640x480. En cuanto a acceso y seguridad, el servidor de video soporta hasta 5 cuentas de usuario distintas para el acceso al video y 5 más para el acceso en modo administrador. Para todas ellas es necesario tener un nombre de usuario y contraseña configuradas. La conexión con la aplicación se lleva a cabo mediante cable cruzado y a través de un switch. El servidor de video está específicamente diseñado para ello y cuenta con un conector de tipo ethernet (RJ45) a tal efecto. Además de físicamente, también es posible la conexión sin cables ya que el servidor soporta el uso de una tarjeta

wireless en formato PCMCIA. El resto de características del servidor así como sus instrucciones de uso pueden consultarse en los documentos adjuntos a este documento.

4.7.3 Aspecto final

El aspecto de la interfaz del servidor de vídeo en modo visualización es el siguiente:



4.8 Program

4.8.1 Descripción

Es la clase por la que comienza a ejecutarse el programa. Contiene el método Main, que marca la línea de ejecución del programa. No es visible en las aplicaciones de tipo Windows, pero todo programa creado en C# contiene un método Main. En este caso el método Main lanza la aplicación, con la ayuda de la clase *Application*, que contiene métodos para configurar y ejecutar el programa. La clase *Application* también puede usarse para detener programas o subprocesos en ejecución o para lanzar bucles de mensajes, que son rutinas que procesan eventos de usuario, tales como clic del ratón o pulsaciones de teclas.

Además de la clase *program*, en este archivo se encuentran dos de las clases de las que ya se ha hablado previamente como son *guardarnombres* y *FTPFactory*. A lo largo de este documento se ha analizado a fondo el funcionamiento de cada uno de los miembros de la clase guardarnombres en los puntos del programa en los que estaban presentes. Por ejemplo el método nombres se aplicaba en la ventana de configuración con la finalidad de leer los nombres de las entradas, el método reconocimiento que se aplicaba en la misma ventana y se utilizaba para recuperar los parámetros de configuración de los módulos. En la ventana de configuración se hacía uso de otros dos métodos pertenecientes a esta clase, como son descargar y actualización, que como sus nombres indican, son utilizados para descargar la configuración actual del módulo y para cargar una configuración modificada en el e.reader respectivamente.

4.8.2 Diseño

4.8.2.1 La clase FTPFactory

La clase FTPFactory proviene de una biblioteca de clases online, por lo que es muy extensa. Dado que muchos de los métodos que ella contiene no serán utilizados, no se ha creído conveniente analizarlos todos, sino solo los más importantes. La clase comienza poniendo unos valores por defecto a la dirección del servidor, el nombre de usuario, la contraseña y el puerto al que se ha de acceder.

El método setRemoteHost simplemente se utiliza para guardar la dirección IP del servidor al que se accederá. Es el primer paso que se dará en el proceso de conexión. Los siguientes pasos también siguen esta misma línea, ya que los métodos setRemoteUser y setRemotePass tienen la misma función con la diferencia de que lo que se almacena es el nombre de usuario y la contraseña. Estos tres métodos esperan recibir datos de tipo string con un formato determinado en el caso de la dirección IP. También se incluye en el proceso de conexión el método setDebug, que recibirá valores booleanos (verdadero o falso). Si Debug está situado en verdadero, se enviarán el nombre de usuario y contraseña. Si es falso significará que no es necesaria autentificación por parte del usuario.

Una vez se han preconfigurado los valores para estas variables, se encuentran las definiciones de los métodos en los que se interactúa con el servidor. Los más relevantes son Login, getFileList, Download(y sus variantes), Upload (y sus variantes) y Close. Estos son los métodos que permitirán llevar a cabo las operaciones más importantes y por ello serán analizados.

4.8.2.2 Método Login

El método más importante que contiene la clase es el método Login. Este método permite la conexión mediante protocolo FTP a un servidor remoto. El primer paso del proceso de conexión es crear un objeto de la clase socket. Para ello se utiliza las sentencias:

```
        clientSocket = new
Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
```

Como puede verse, se ha de especificar el tipo de protocolo de comunicación que se utilizará con este objeto, siendo en este caso el protocolo TCP, ya que FTP está contenido en él. También es necesario crear un objeto de IPEndPoint, que consta de dos valores: La dirección IP del servidor y el número de puerto asociado a ella. Este objeto es conocido como socket en nomenclatura de redes, pero no debe ser confundido con la clase del mismo nombre. Para lanzar la conexión se utiliza el método connect de la clase socket y se le envía el objeto IPEndPoint creado. De esta manera, se lleva a cabo el three way handshake típico del protocolo FTP. Como es lógico el método tiene previstos los casos en los que no se pueda realizar la conexión o exista algún fallo. En este caso concreto se lanzará una excepción que parará la ejecución del programa y mostrará un mensaje informando de que no ha podido establecerse la comunicación con el servidor. Una vez terminada la conexión lee la respuesta del servidor, que deberá ser la 220 (servidor preparado) para que la conexión siga adelante.

El segundo paso de la conexión es enviar el nombre de usuario y la contraseña para la autentificación. En primer lugar se enviará el nombre de usuario utilizando el método sendCommand. Si la respuesta del servidor refleja que el nombre de usuario es correcto se pasará a enviar la contraseña, en caso contrario la conexión fallará. Se repite el proceso para la contraseña. La cláusula if con la que se activan las excepciones es la siguiente:

```
if (!(retValue == 331 || retValue == 230))
{
    cleanup();
    throw new IOException(reply.Substring(4));
}
```

Cuando el nombre de usuario y la contraseña han sido aceptados se puede considerar que la conexión con el servidor está abierta, por lo tanto se procede a indicar tal estado en la variable booleana logined. Usando esta variable se evitará que se intente iniciar una conexión cuando hay otra activa.

Si la aplicación es de tipo consola el usuario recibirá el mensaje de que la conexión ha sido realizada con éxito. Por último, se guarda el nuevo path del servidor para que en caso de que se desee enviar algún archivo el usuario sepa a que directorio del servidor se está enviando. El método utilizado para esto es chdir y también muestra un mensaje visible para aplicaciones de consola. De esta manera termina el método login, resultando en una comunicación abierta entre la aplicación y el servidor del e.reader.

4.8.2.3 Método getFileList

Otro de los métodos de ha sido utilizados en el programa es getFileList. La función de este es llevar a cabo una petición de envío del listado de archivos contenidos en el servidor. Para ello, en primer lugar, se comprueba si hay una comunicación abierta con el servidor (con el valor de logined) y en caso de que no la haya se conecta. Para hacer la petición del listado no hay más que enviar un comando list con la mascara correspondiente, para ello utiliza el método sendCommand, que se verá reflejado en la comunicación por un comando LIST. La máscara informa de la estructura de texto que se va a recibir. En este caso se trata de un nombre seguido de un punto y tras este una terminación que también será de tipo texto. La representación de esta estructura en forma de máscara es un asterisco seguido de un punto y de otro asterisco (*.*). La recepción de los datos es algo compleja ya que se trata de un stream de bytes y no de un archivo, por lo que ha de ayudarse del método Receive de la clase Socket para ello. Acto seguido, convierte los bytes recibidos a formato ASCII, introduce un salto de línea entre cada nombre de archivo, guarda cada uno en una posición de un array de texto y cierra la comunicación. Por último, devuelve al programa el array con los nombres de los archivos. Este método es muy útil cuando no se conocen los archivos a descargar, o como en este caso, cuando se sabe la estructura de los nombres pero no el número de archivos que existen.

4.8.2.4 Método Download

Uno de los métodos básicos dentro de la clase es el utilizado para descargar archivos del servidor remoto. Su nombre es download. En realidad existen cuatro métodos con este mismo nombre y que difieren, como se explicará a continuación, en algunas de sus funcionalidades aunque todos parten de un método original. El primer caso del método download tan solo necesita que se le especifique el archivo que se desea descargar en formato de string. La petición de descarga se lleva a cabo con la llamada al método download original que se verá más adelante y se ve reflejada por el comando RETR en la comunicación. Aquí puede verse la sentencia, donde remFileName es el nombre del archivo a descargar:

```
download(remFileName, "", false);
```

Es el método download más sencillo, ya que simplemente descargará el archivo en el directorio de la aplicación y mantendrá el nombre del archivo original. Este será el utilizado en la aplicación ya que no se necesitan las funcionalidades algo más avanzadas del resto de casos. El segundo caso comparte la misma sentencia con el anterior con una

única diferencia. El tercer parámetro se sustituye por *resume*. Esto hace que el flag de *resume* se active, se verá el efecto de este indicador más adelante.

El tercer caso también es similar a los anteriores, pero introduce una variante, y es que se debe especificar un nuevo nombre para el archivo, e incluso se puede indicar el directorio a donde se desea que se realice la descarga. En este caso el método necesita dos datos. Por un lado el nombre del archivo a descargar y por otro el nuevo nombre y la dirección donde se guardará. Si el nombre con el que se va a guardar el archivo ya existe se sobrescribirá, sin embargo si el fichero donde se desea realizar la descarga no existe esta fallará, por lo que es muy importante que el path especificado sea correcto.

El último caso del método download aúna todos los anteriores, ya que es el método original y los otros son casos especiales simplificados. Comienza comprobando si existe una comunicación activa y en caso negativo la crea. Dado que va a llevarse a cabo una transferencia de archivos es necesario que el servidor se encuentre en modo binario, por lo que se utiliza el método SetBinaryMode y se configura como verdadero. Si no consta el nombre con el que se quiere guardar el archivo descargado, el propio programa le da el valor del nombre original del archivo y si este archivo no existe se crea. Una vez hecho esto se realiza la petición de descarga mediante la línea:

```
sendCommand("RETR " + remFileName);
```

Tras esto, el programa se prepara para recibir el archivo con la ayuda de un búfer, como se hacía en el método en el que se descargaba el listado de archivos del servidor. EN este caso la novedad radica en que se ha de escribir la información recogida en un archivo, por lo que se utiliza el método write. Como en el caso anterior, aquí también están previstos los errores en la comunicación, de los que se informa mediante excepciones. Una vez finalizada la descarga, se cierra el método de escritura y la comunicación, pero no la conexión con el servidor, ya que es habitual realizar más de una acción por lo que el cierre de la conexión obligaría a perder mucho tiempo.

4.8.2.5 Método Upload

El método upload sirve para cargar archivos en el servidor. Ocurre lo mismo que en el caso de las descargas y es que existen dos definiciones para el mismo método, una simplificada, que será la que se usará en este caso, y el método original. La definición simplificada simplemente carga en el servidor un archivo con el nombre que se le proporcione. Le método original añade a esto el flag de resume.

Este método sigue los mismos pasos que download hasta que llega el punto en el que se ha de cargar el archivo. Para llevar a cabo esta acción se implementa el comando STOR acompañado del nombre y directorio del archivo, como puede verse en la sentencia siguiente:

```
sendCommand("STOR " + Path.GetFileName(fileName));
```

Como puede verse, no es necesario introducir el path manualmente, sino que proporcionando simplemente el nombre del archivo el propio programa encuentra su localización. Para enviar la información del archivo, se utiliza el método de lectura de cadenas de texto y una vez almacenados los datos en el búfer se envían con el comando Send, a diferencia de los casos anteriores, en que se utilizaba sendCommand. La diferencia entre estos dos métodos es que aunque ambos se utilizan para enviar información, esta es de dos tipos distintos. sendCommand se utiliza en caso de que la información a enviar sea un comando, es decir para notificaciones o peticiones (RETR, STOR, etc.). Sin embargo, cuando lo que se desea enviar es una cadena de bytes (de contenido diverso) no puede utilizarse sendCommand (ya que la información no es un comando reconocible ni es fija) y por ello se utiliza el método general Send. En aplicaciones de consola, se podrá ver un mensaje que informa del estado de la descarga (si está iniciándose, en curso o ha finalizado) así como los directorios de origen y destino. Una vez finalizado el envío se cierra la instancia el lector de streams y la comunicación.

4.8.2.6 Método Close

Otro de los métodos más útiles de la clase es el de cierre de la conexión. En esta aplicación se ha decidido no usarlo debido a que el carácter modular del programa lo hace innecesario. Al cerrar la ventana en la que se tiene una conexión activa esta se cerrará automáticamente. El método close es muy sencillo ya que simplemente consta de una sentencia para evaluar si hay alguna conexión abierta y de otra para cerrarla en caso afirmativo. Tiene la forma siguiente:

```
if (clientSocket != null)
{
    sendCommand("QUIT");
}
```

La condición evalúa el estado del objeto clientSocket derivado de la clase Socket, que como se ha visto anteriormente sirve para realizar las conexiones al servidor. Como se pudo comprobar al inicio del método login, a los objetos de la clase socket es necesario entregarles unos parámetros para poder operar con ellos. Si el valor del objeto no es nulo, este está preparado para realizar sus funciones y es posible que haya una conexión abierta. Por lo tanto, el método enviará el comando QUIT al servidor, que iniciará el proceso de desconexión. Además de los que se han analizado, en la clase FTPFactory están contenidos numerosos otros métodos que permiten implementar cualquier funcionalidad deseable al trabajar con un servidor FTP, como por ejemplo cambios de modo (binario/ASCII), renombrar archivos almacenados en el servidor, abrir carpetas en el servidor, borrar archivos o ver su tamaño, entre otras acciones.

5 Conclusiones

La aplicación resultante del proyecto cumple las expectativas y objetivos que se marcaron al inicio del mismo, ya que es capaz de comunicarse con los módulos, configurar las entradas de los mismos, recibir datos de los sensores y presentar toda esta información de una manera clara e intuitiva. Además, se han añadido funcionalidades como la recepción de vídeo y la representación gráfica histórica y se han agilizado los procesos de configuración y descarga de datos. Sumado a todo esto está la facilidad para actualizar y personalizar la interfaz por cualquier programador. Por todo ello, se puede considerar que la interfaz mejora el programa de control ofrecido por el fabricante de los módulos.

Dado el dinamismo del mundo de la domótica en el que día tras día se dan nuevos avances y aparecen nuevas tecnologías, no se debe considerar la aplicación como cerrada, sino como una base a la que pueden añadirse nuevas funcionalidades como la activación de alarmas y actuadores, el envío de avisos por email o sms y otras muchas que sin duda surgirán en el futuro.

Pamplona, Septiembre de 2010

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

Elaboración de una interfaz de comunicación para módulos domóticos

DIAGRAMAS

Asier Ortés Alfonso

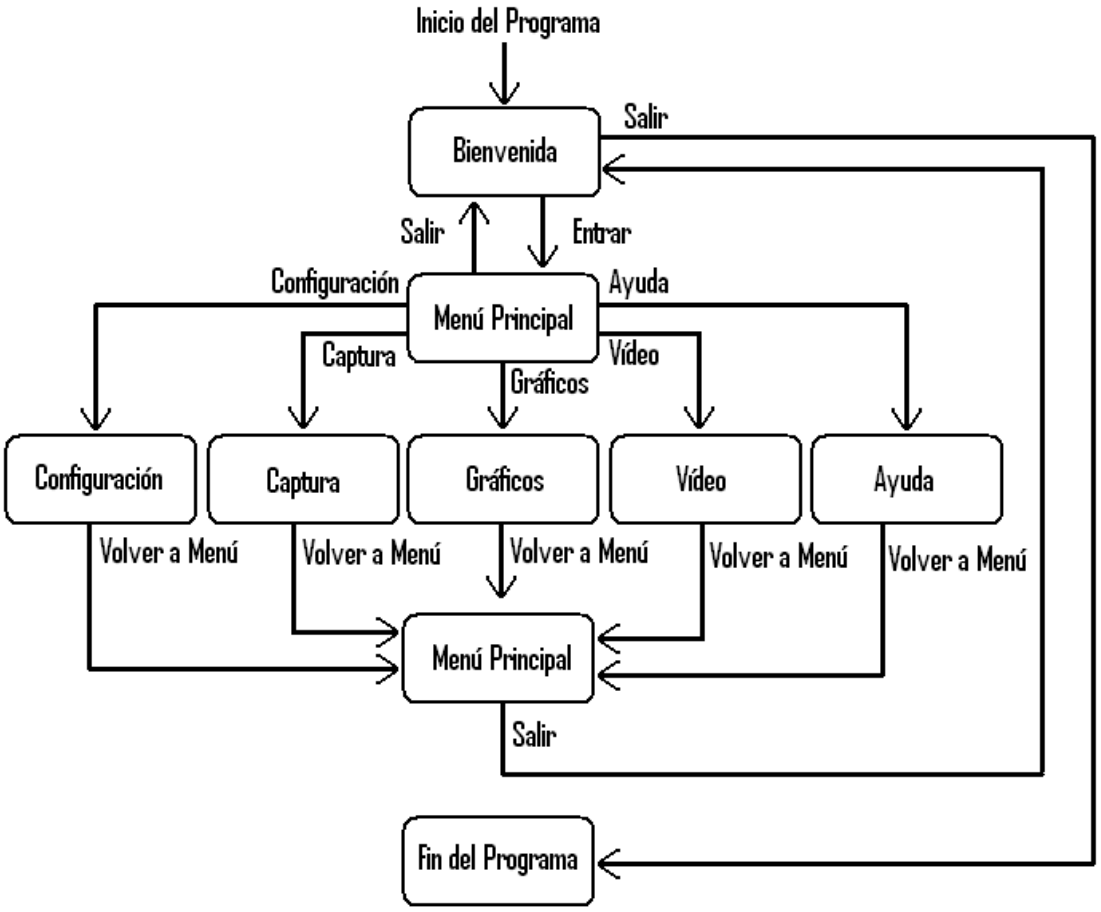
Tutor: Ignacio Raúl Matías Maestro

Pamplona, Septiembre de 2010

Índice

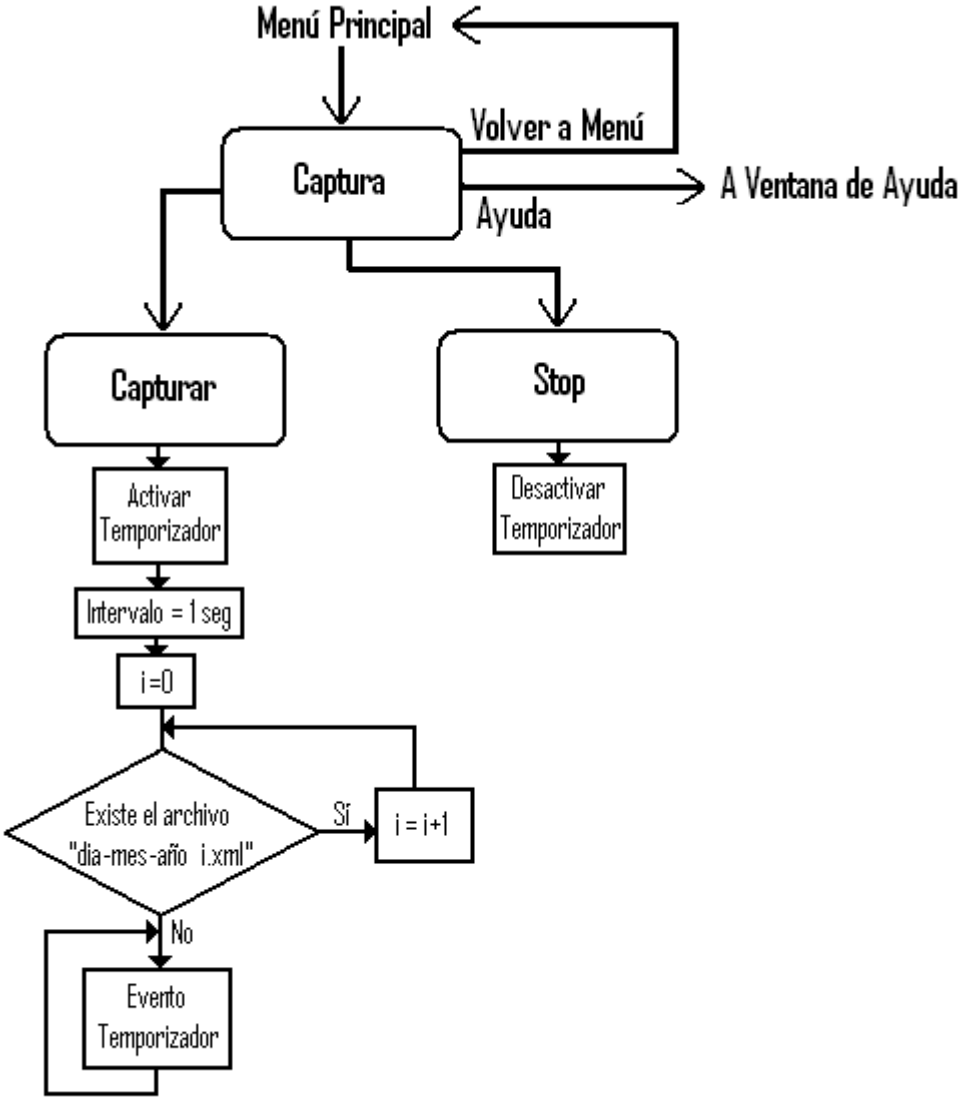
1. Diagrama general de la aplicación 102
2. Captura 103
    2.1 Diagrama ventana de captura 103
    2.2 Diagrama de evento temporizador 104
3. Configuración 105
    3.1 Diagrama general de configuración 105
    3.2 Actualizar 106
    3.3 Descargar 107
    3.4 Selección módulo 108
4. Gráficos 109
    4.1 Diagrama general de gráficos 109
    4.2 Método crear gráfica 110

1. Diagrama general de la aplicación

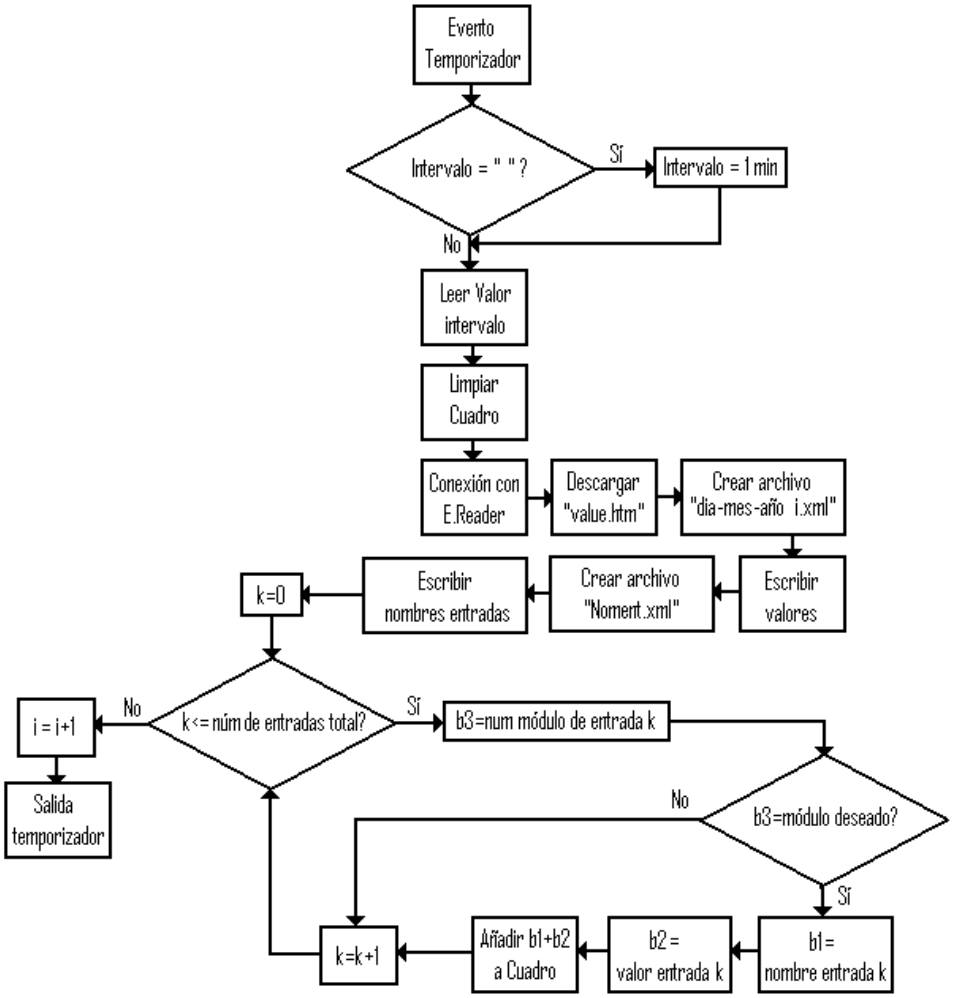


2. Captura

2.1 Diagrama de ventana de captura

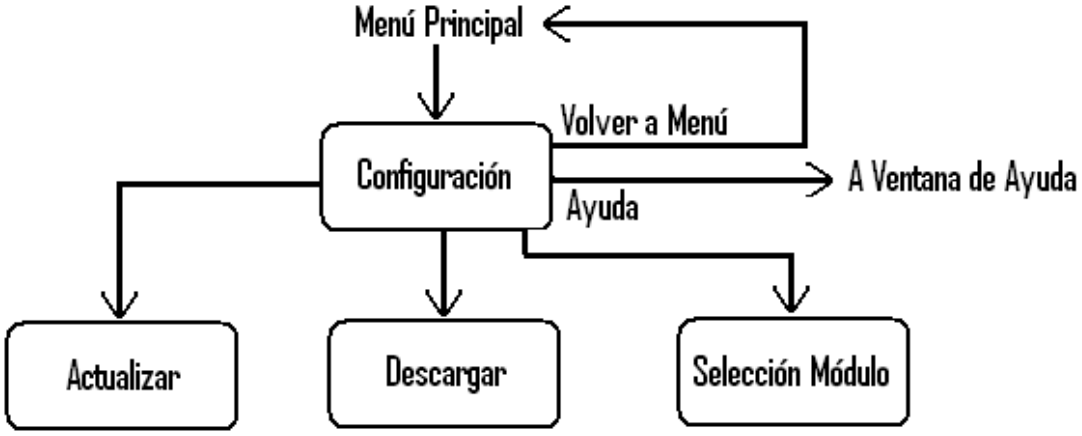


2.2 Diagrama de evento temporizador

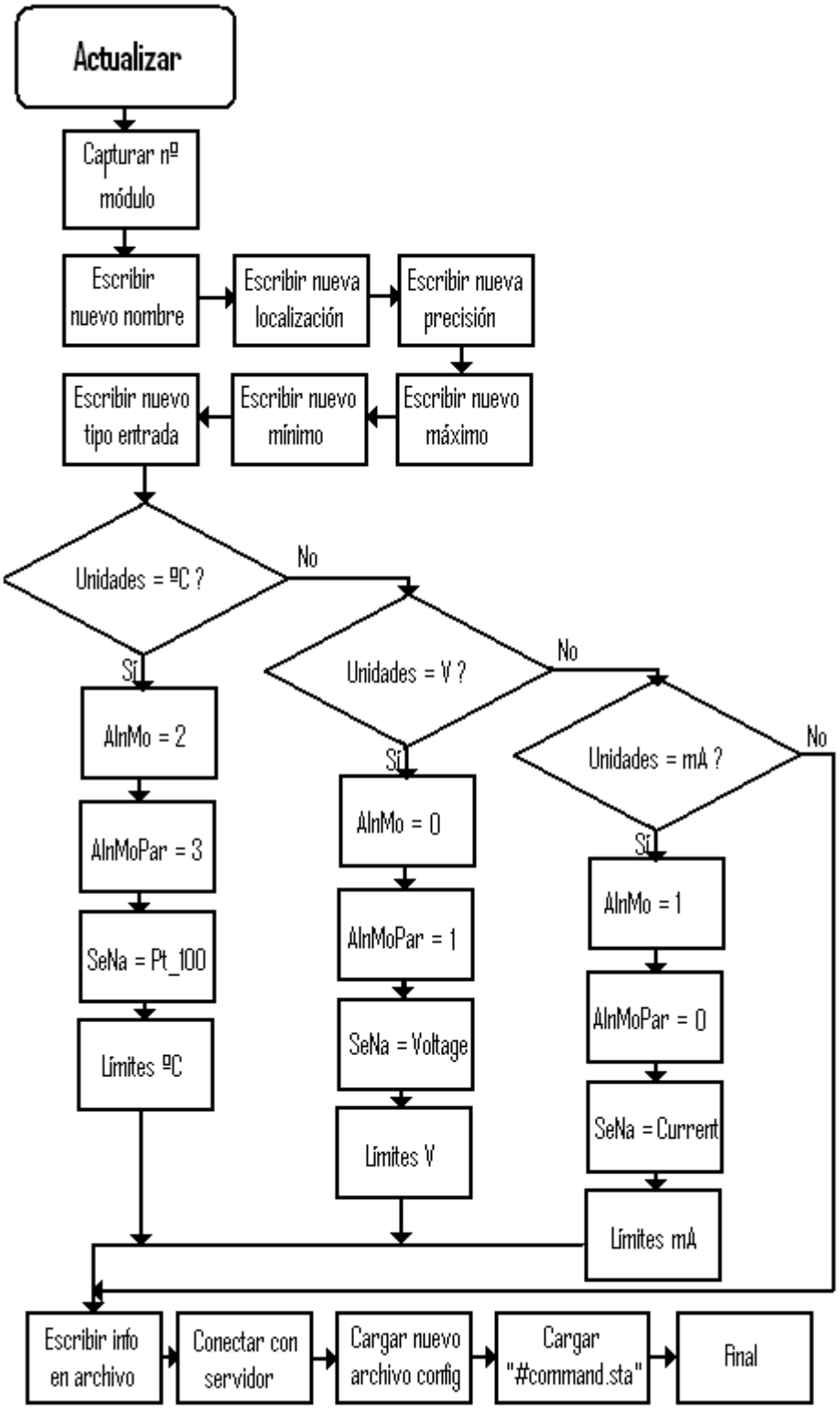


3. Configuración

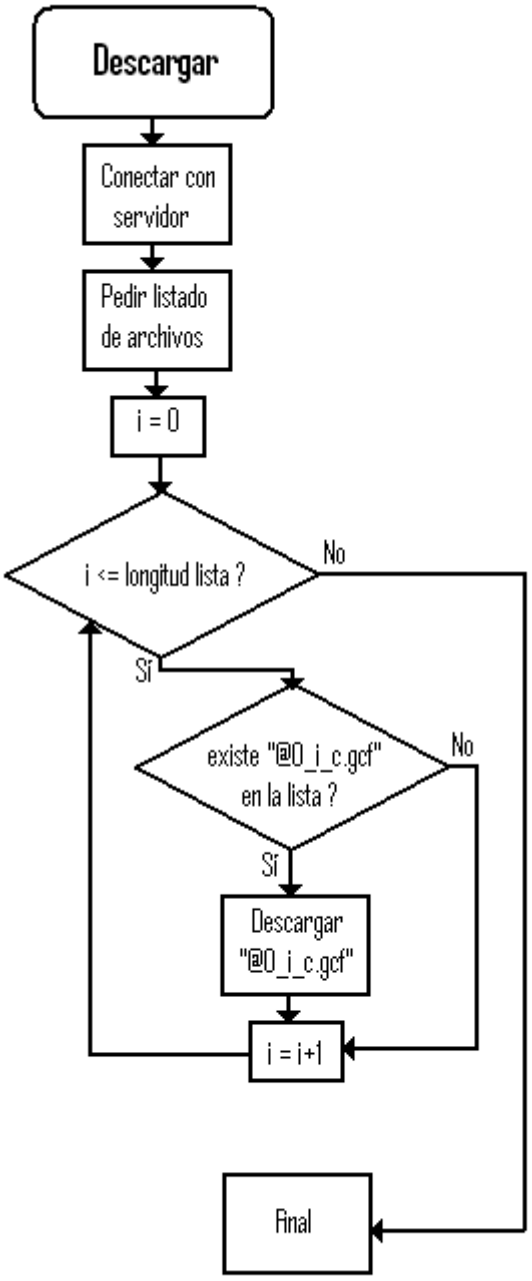
3.1 Diagrama general de configuración



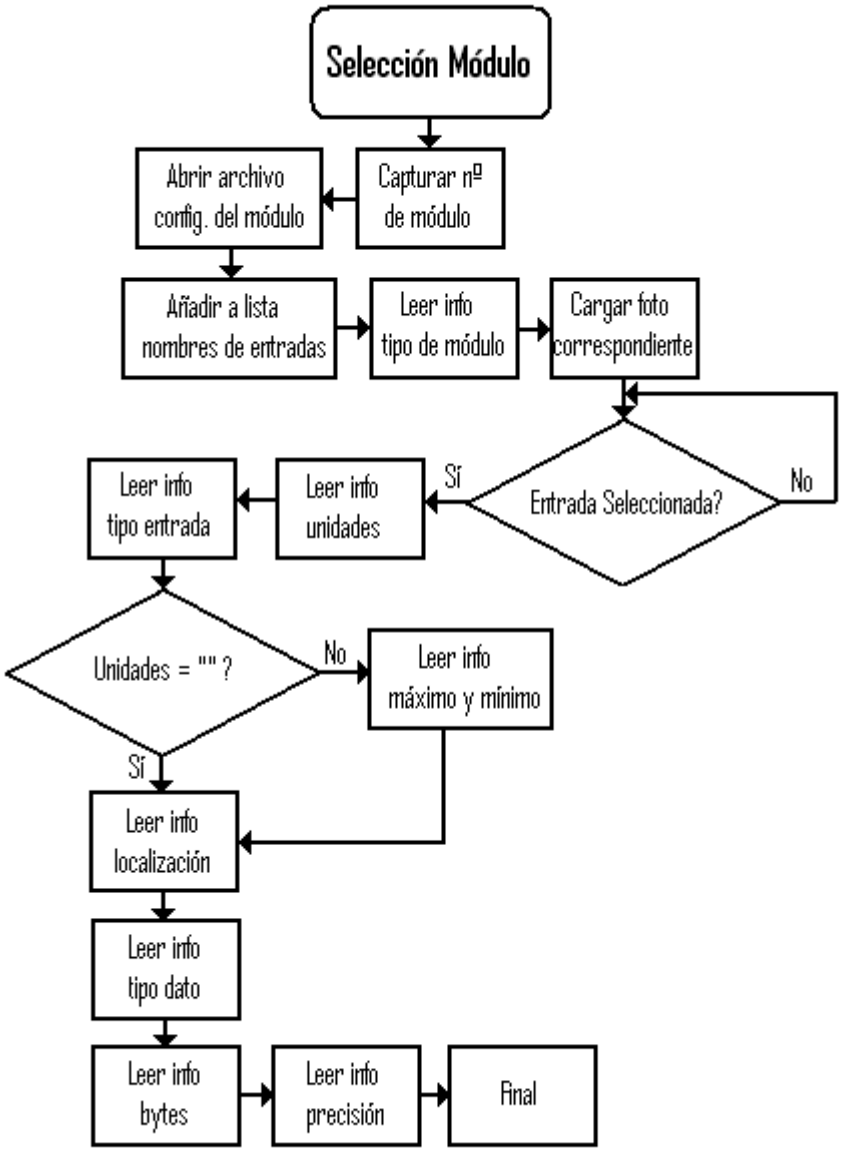
3.2 Actualizar



3.3 Descargar

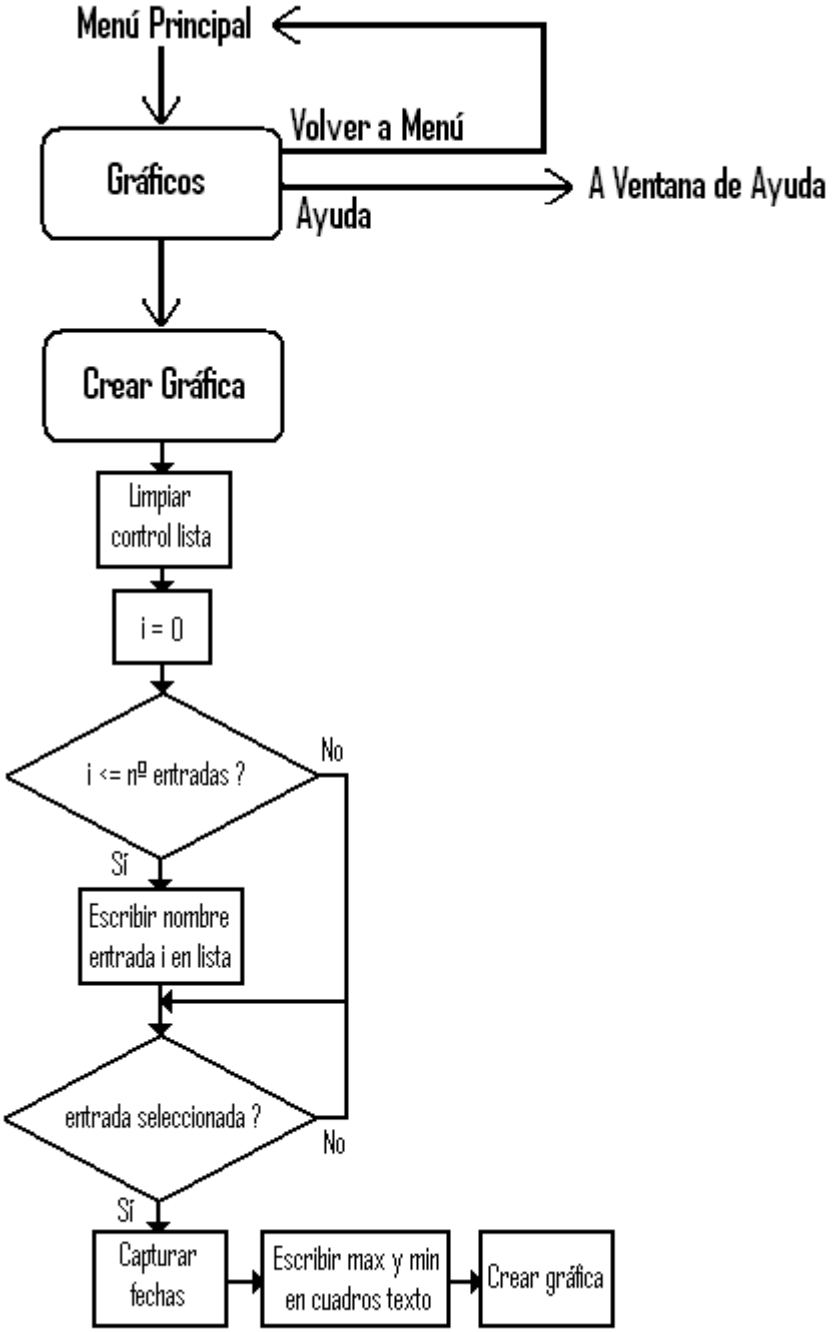


3.4 Selección Módulo

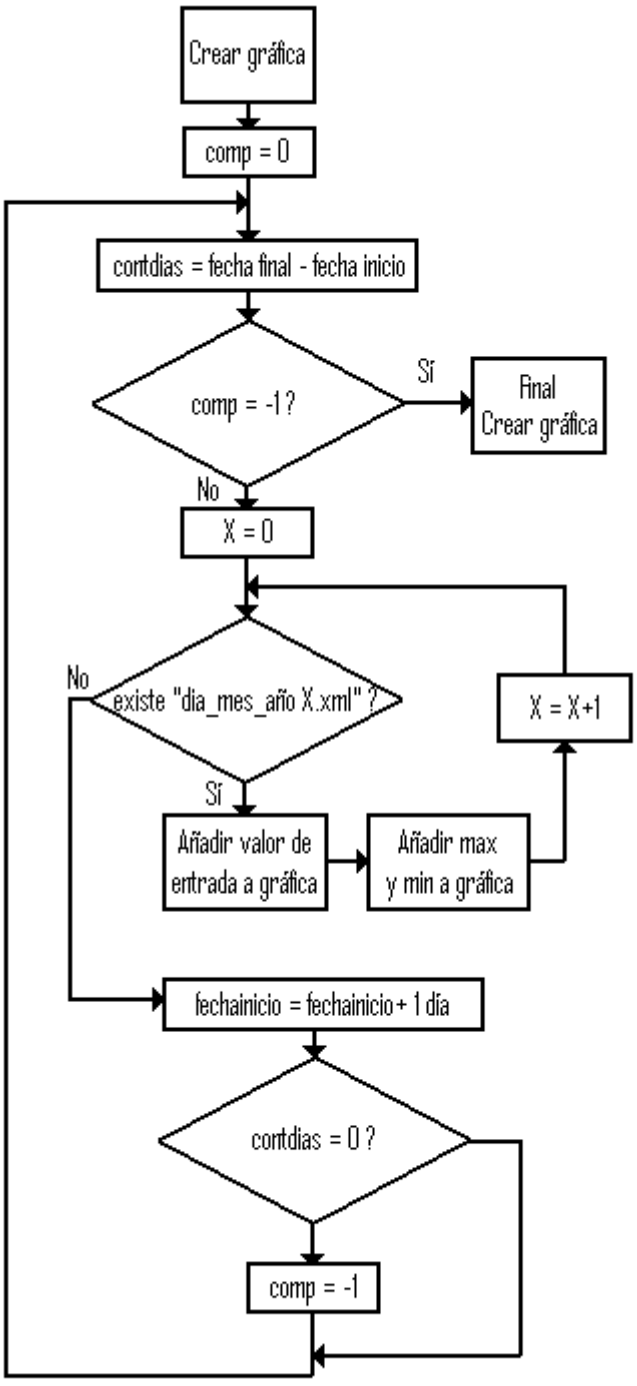


4. Gráficos

4.1 Diagrama general de gráficos



4.2 Método Crear gráfica



ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

Elaboración de una interfaz de comunicación para módulos domóticos

PLIEGO DE CONDICIONES

Asier Ortés Alfonso

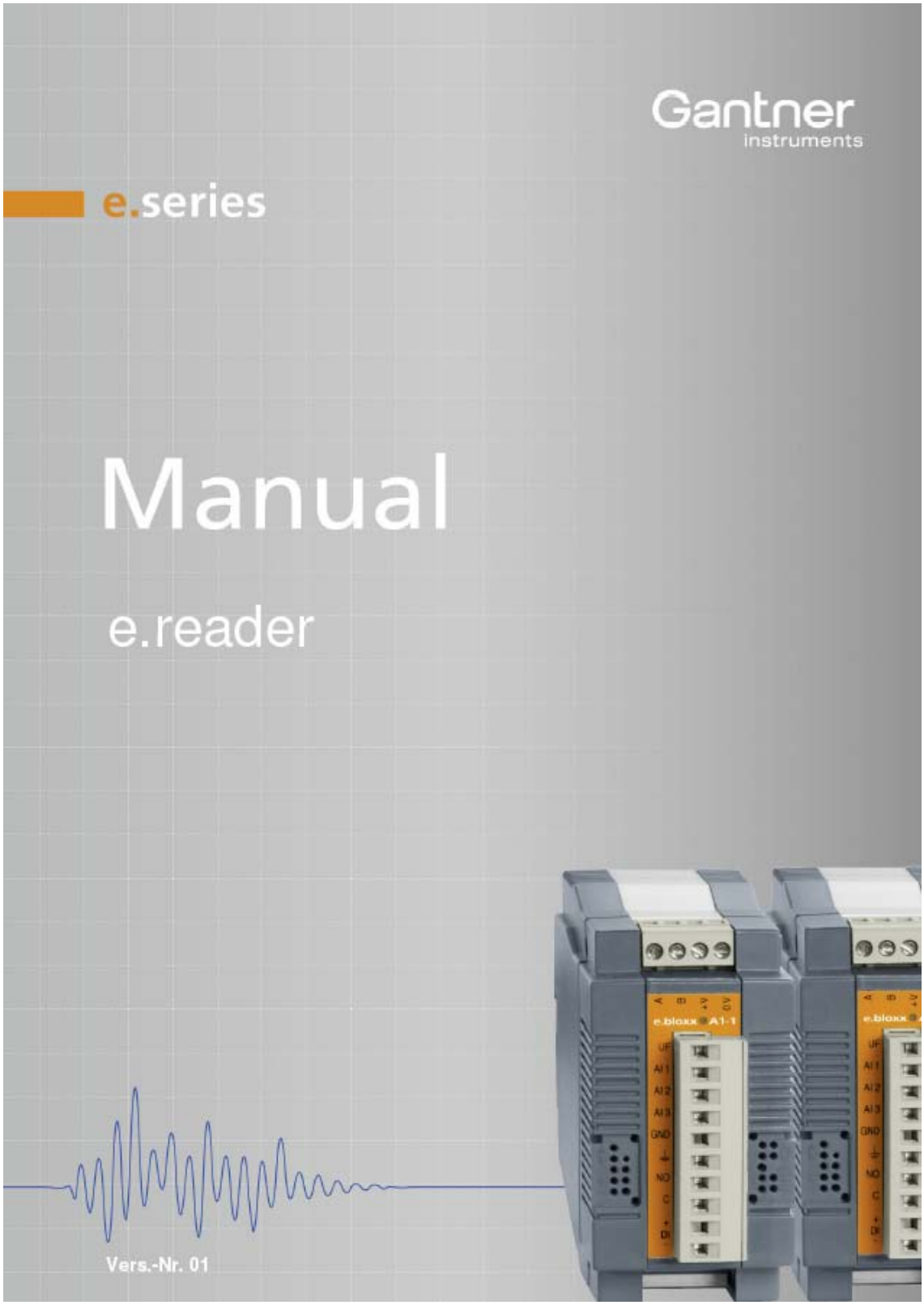
Tutor: Ignacio Raúl Matías Maestro

Pamplona, Septiembre de 2010

Índice

1. Módulo de inteligencia E.Reader 113
2. Módulo de entradas/salidas E.Bloxx A1-8 187
3. Módulo de entradas/salidas E.Bloxx D2-1 226
4. Servidor de vídeo y Webcam 251
5. Protocolo TCP/IP 284
6. Protocolo FTP 285
7. Especificación de C# 286

1. Módulo de inteligencia E.Reader



Important: Before commencing installation, commissioning, putting into service and before any maintenance work is carried out, it is essential that the relevant warning and safety instructions in this manual are read!

General warning and safety instructions:

Dear customer,

We congratulate you on having selected a product of Gantner Instruments Test & Measurement GMBH. So that our product functions in your installation with safety and to your complete satisfaction, we take this opportunity to familiarize you with the following ground rules:

1. Installation, commissioning, operation and maintenance of the product purchased must be carried out in accordance with instructions, i.e. in accordance with the technical conditions of operation, as described in the corresponding product documentation.

2. Before installation, commissioning, operation or maintenance it is therefore essential that you read the corresponding chapter of this manual and observe its instructions.

3. If there are still some points on which you are not entirely clear, please do not take a chance, but ask the customer adviser responsible for you, or ring the Gantner Instruments Test & Measurement GMBH hot line.

4. Where not otherwise specifically laid down, appropriate installation, commissioning, operation and maintenance of the appliance is the customer's responsibility.

5. Directly on receipt of the goods, inspect both the packaging and the appliance itself for any signs of damage. Also check that the delivery is complete (-> accessories, documentation, auxiliary devices, etc.).

6. If the packaging has been damaged in transport or should you suspect that it has been damaged or that it may have a fault, the appliance must not be put into service. In this case, contact your customer advisor. He will make every effort to resolve the problem as quickly as possible.

7. Installation, commissioning and servicing of our appliances must only be carried out by suitably trained personnel. In particular, correspondingly qualified specialists may only make electrical connections. Here, the appropriate installation provisions in accordance with the relative national Electrical Engineers construction regulations (e.g. ÖVE, [Austrian] VDE, [German]...) must be observed.

8. Where not otherwise stated, installation and maintenance work on our appliances is exclusively to be carried out when disconnected from the power supply. This applies in particular to appliances that are normally supplied by low-tension current.

9. It is prohibited to make alterations to the appliances or to remove protective shields and covers.

10. Do not attempt yourself to repair an appliance after a defect, failure or damage, or to put it back into operation again. In such cases, it is essential you contact either your customer adviser or the Gantner Instruments Test & Measurement GMBH hot line. We will make every effort to resolve the problem as quickly as possible.

11. Gantner Instruments Test & Measurement GMBH accepts no responsibility for any injuries or damage caused as a result of improper use.

12. Although every care is taken and we are continuously aiming for improvement, we cannot exclude completely the possibility of errors appearing in our documentation. Gantner Instruments Test & Measurement GMBH therefore accepts no responsibility for the completeness or the accuracy of this manual. The right is reserved to make alterations, and we may carry out alterations at any time without giving prior notice.

13. Should you discover any fault with the product or in its accompanying documentation, or have any suggestions for improvement, you may confidently approach either your customer adviser or Gantner Instruments Test & Measurement GMBH directly.

14. However, even if you just want to tell us that everything has functioned perfectly, we still look forward to hearing from you.

We wish you a successful application of our appliances. We will be pleased to welcome you as a customer again soon.

Contact address / manufacturer:

Gantner Instruments Test & Measurement GmbH
Montafonerstrasse 8
A - 6780 Schruns/Austria
Tel.: +43 5556 73784 - 410
Fax: +43 5556 73784 - 419
E-Mail: office@gantner-instruments.com
Web: www.gantner-insturments.com

Gantner Instruments Test & Measurement GmbH
Industriestraße 12
D-64297 Darmstadt
Tel.: +49 6151 95136 - 0
Fax: +49 6151 95136 - 26
E-Mail: testing@gantner-instruments.com
Web: www.gantner-instruments.com

TABLE OF CONTENTS

1. ABOUT THIS MANUAL 6
2. SYSTEM DESCRIPTION 7
3. START-UP 8
3.1 Supply and Connection of the e.reader 8
3.2 Installation of the Configuration Software "test.commander" 9
3.3 Network Scan for connected controller 10
3.4 Traditional way of finding an e.reader on the Ethernet 11
3.5 Read Configuration from an e.reader 13
3.6 Edit a Configuration 13
3.6.1 Configuration of e.reader 13
3.6.2 Configuration of Variables 14
3.7 Sensor Connection 15
3.8 Display Measurement Values 15
3.9 Read Memory 16
3.10 Test.con (optional) 17
4. SENSOR CONNECTION 18
4.1 General 18
4.2 Analog Input 18
4.2.1 Measurement of Voltage 18
4.2.2 Measurement of Current 19
4.2.3 Measurement a Resistance 20
4.3 Digital Inputs / Outputs 21
4.3.1 Digital Inputs 21
4.3.2 Digital Outputs 21
4.4 Relay Output 22
5. CONFIGURATION 23
5.1 Open a Project 23
5.2 Adding an e.reader to a Project 25
5.3 General controller / e.reader settings 26
5.3.1 Slave Interface 26
5.3.2 Host Interface 27
5.3.3 e.reader Settings 28
5.3.4 Low-Power-Functionality- Power down mode 29
5.3.5 Low-Power-Functionality- Modem Communication 30
5.4 Configuration of the Variables of an e.reader 31
5.4.1 Variable Type 31
5.4.2 Sensor Name 32
5.4.3 Scaling Type 34
5.4.4 Error Handling 34
5.4.5 Data Format Settings 34
5.5 Configuration of Communication 35
5.5.1 E-mail 36
5.5.2 FTP - Client 38
5.5.3 PPP – Connection for modem communication 40
5.5.4 Truecon – modem with firewall 43

5.5.5 (A)DSL modem 43
5.5.6 USB- Data Transfer 44
5.6 Data File Management 45
5.7 Adding e.bloxx modules to an e.reader 46
5.7.1 Configuration of e.bloxx variables 47
5.7.2 Limitation of additional e.bloxx modules 47
5.8 Statistics 48
5.8.1 Design Rule Check 49
5.9 Display of Slave Information 49
5.10 Display of Status Information from Concentrator 50

6. FUNCTIONALITY WITH TEST.CON 51
6.1 Overview of the test.con User Interface 51
6.2 Projects 52
6.3 Classes and Instances 53
6.4 The Four States of the test.con Software 53
6.4.1 Edit State 53
6.4.2 Design State 55
6.4.3 Run State 56
6.4.4 Online Observation State 57
6.5 Starting a Project 57
6.6 Connecting e.reader and test.con 59
6.7 Practical Use of the Structuring 60

7. HARDWARE INSTALLATION 63
7.1 Environmental Conditions 63
7.2 Mounting 63
7.3 Module Parts 63
7.3.1 Front LEDs 64
7.4 Connection Technique 64
7.5 Power Supply 64
7.6 Connection of e.bloxx Modules 65
7.7 Connection to Ethernet 66
7.8 Connection to the RS 485 Communication Bus 66
7.9 Shielding RS 485 67
7.10 LED Indication 68
7.10.1 Indication of RUN and ERROR States 68

8. SPECIFICATIONS 70
8.1 Analog Inputs (8 per module) 70
8.2 Analog/Digital Conversion 70
8.3 Digital In- and Output (6 per module) 71
8.4 Relais Outputs (2 per module) 71
8.5 Data Memory 71
8.6 Interfaces 71
8.7 Power Supply 72
8.8 Mechanical 72
8.9 Connection 72
8.10 Environmental Conditions 72
8.11 Electromagnetic Compatibility 72

1. ABOUT THIS MANUAL

This manual describes the installation and setup of the **Data Acquisition and Logging Module e.reader** from Gantner Instruments Test & Measurement GmbH. The e.reader module is a data logging module with integrated measurement capabilities.

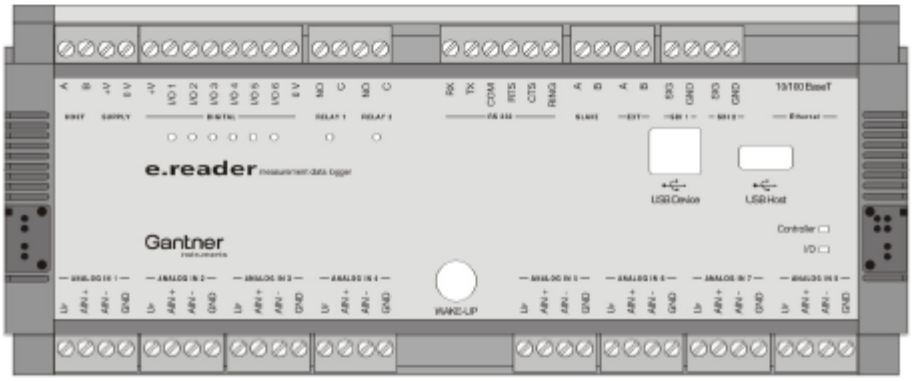
The following information can be found in this manual:

- Description of the e.reader system with detailed information on the hardware and module features.
- A start-up guide which describes how the e.reader is put into operation.
- Installation description of the e.reader and how the module is connected to the power supply and bus lines.
- Description of the different types of measurement of the e.reader.
- An introduction on how the e.reader modules are configured with the Configuration Software test.commander. This software has an integrated help including a detailed description of the configuration process.
- Introduction into the software test.con, which is used to define the functionality of an e.reader.
- Technical specifications of the e.reader.

2. SYSTEM DESCRIPTION

The ***Data Acquisition Module e.reader*** is a Data Logger, used for the industrial and experimental testing technology, especially for the multi-channel measurement of electrical signals of thermal or mechanical data at test beds and test sites.

The e.reader provides the advantages of a remote structure and at the same time high precision, dynamic and flexible PAC (programmable automation controller) functionality. Eight analog inputs as well as the possibility to define conditioning, combinations, control functionalities, sequencing, mathematical calculations and data logging independently are available. The unit provides six configurable digital in- and outputs for thresholds, alarms, taring, trigger, and two relay outputs as well as an 128 MByte data memory, which can be divided into different memory depths, logging rates and trigger conditions. Additionally this module has an RS485 slave interface to connect additional e.reader E (expansion module) or e.bloxx modules and an Ethernet interface.



Picture 2.1 - Front View of the e.reader Module

The following instruction will show the simple setup of an e.reader system:

1. Installation, chapter 3.1
2. TCP/IP-Setup, chapter 3.4
3. Sensor connection and configuration, chapter 4
4. Read online measurement values, chapter 3.8 and measurement buffer, chapter 3.9
5. test.con application (optional), chapter 3.10

For several applications the system will be connected to a superior system. Therefore the e.reader provides several interfaces such as the Modbus TCP/IP, USB slave, USB master and it can be connected to any SCADA package using one of the standardised interfaces.

For more details on the test.commander software please refer to chapter 3.2

3. START-UP

This chapter describes how the module e.reader is set into operation, the way it is set-up and which way measurement values can be read by using the configuration software test.commander.
The chapter will just give a short introduction so it is possible to check the correct function of the e.reader module. Detailed information on the installation, set-up, the test.commander and test.con software can be found later in this manual.

In order to work with an e.reader module you need the following parts:

- e.reader module
- Configuration software test.commander
- Automation software test.con (optional)
- RS 232 connection cable ICL106 which is delivered with the e.reader module (if the IP address of the e.reader has to be changed)
- Ethernet cable, cross wire for direct connection to PC or normal one for connection to a network

3.1 Supply and Connection of the e.reader

Use the e.reader module only within the allowed environmental conditions (see technical data). The protective system is IP 20 (not water-proof) and the allowed temperature range for operation is -20 up to +60 °C.

An e.reader module can be connected either to a local network or directly to a PC/laptop via RS 232 and Ethernet. The standard operation is to connect the e.reader to Ethernet for integration into a local network but a modem connection via the RS 232 interface is very common as well.

After connection the e.reader has to be supplied with a DC voltage in the range of +10 to +30 VDC. The power input of the e.reader is protected with an internal fuse (reversible) against excessive voltage and polarity connecting error. The power supply cables are connected to the screw terminals "10..30V" and "0 V".

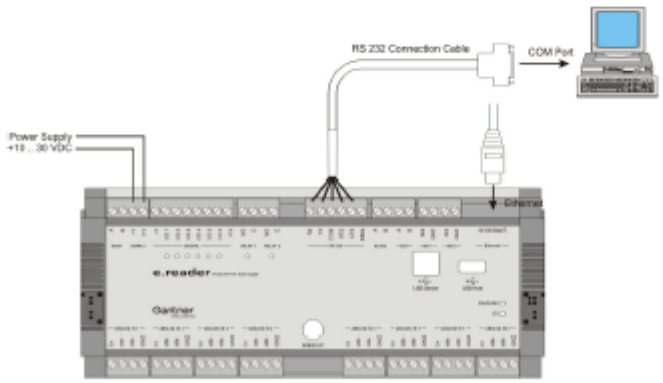


Figure 3.1 - Power Supply and Bus Connection of the e.reader

When turning on the power supply the LEDs of the e.reader will start blinking a few seconds and afterwards the LED "Run/Error" on the front of the e.reader will light up in green color.

3.2 Installation of the Configuration Software "test.commander"

The test.commander software is used to define all settings of an e.reader, like the interface settings and internal settings for the memory (circle buffer), variables (channels), synchronization and life time. This software also includes the Configuration Software ICP 100 for configuration of e.bloxx measurement modules and the visualization tool Green Eye Writer to evaluate measurement data.

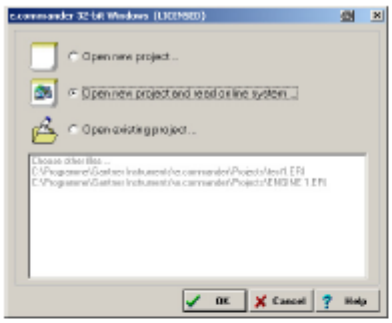
Minimal System Requirements for test.commander

- PC with Windows 98 or higher
- Min. 20 MByte of free disc space to store all projects and history data
- Connected Ethernet Interface

Install the test.commander software from the installation CD. The installer will guide you through the process. Afterwards start the test.commander. An icon will be generated on the desktop and the test.commander can also be found in the Start-Menu of Windows, section "Programs" -> "Gantner Instruments".

At the first start-up of test.commander you have to select the language you want to use. The language can be changed later on if required. Therefore edit the test.commander ".ini" file in the test.commander directory and change the number of the language (0=English, 2=German). Now the test.commander software has to be licensed, otherwise the test.commander will work in demo mode. The language selection and licensing has to be done only once when starting this software the first time.

Now the following window will open (this will be the first window when starting the test.commander next time).



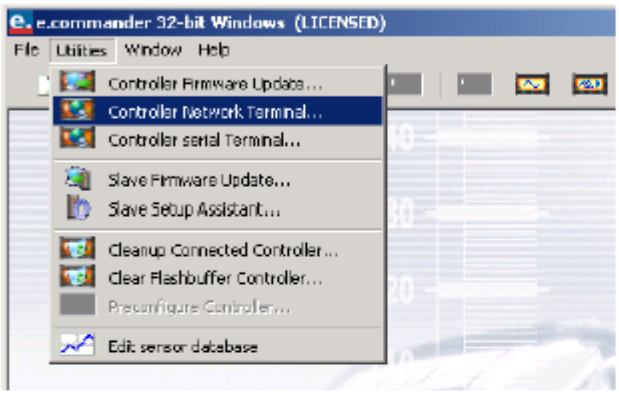
Having connected the e.reader to the local network or to the PC via a cross cable, select the option "Open new project and read online system..." and press "OK". The test.commander will scan the network for connected controllers (see chapter 3.3).

In case you have connected a new e.reader directly to the PC/laptop via the RS 232 interface you can select the option "Open new project" and press the "OK" button to proceed insetting the correct IP address (see chapter 3.4). Otherwise this functionality is being used to define an offline project.

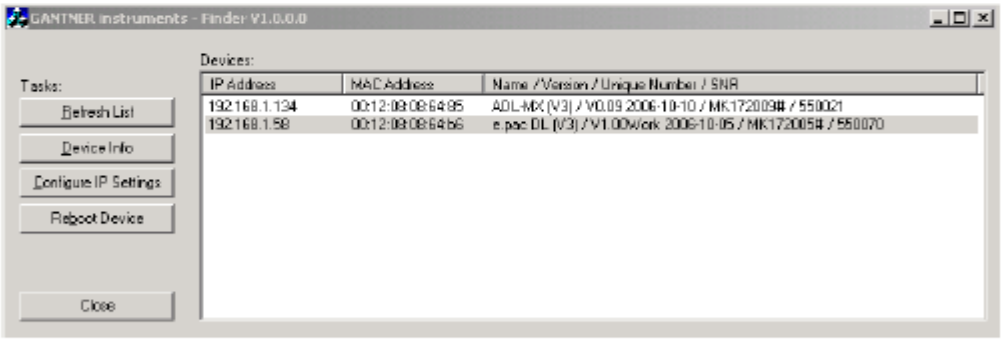
The third possibility (Open existing project...) is being used when changing an existing project offline.

3.3 Network Scan for connected controller

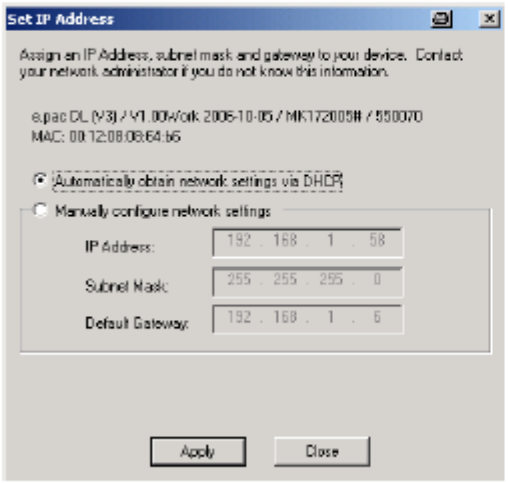
All e.reader units are being delivered with a certain default setting (IP-address). But in some cases it is necessary to handle an e.reader you do not know the settings in detail. Therefore the test.commander provides a very comfortable tool to find the required modules in the network. From the toolbar select Utilities → Controller Network Terminal...



This network scan will find any units being connected to the Ethernet, independent from its IP-Address, its type (e.gate / e.pac / e.reader / e.xact) or if DHCP is activated or not. The scan will provide the following information to be able to adapt the unit for the network it has to be placed into.



To get the required information on the controller / e.reader select the unit and click the button "Configure IP Settings". The window opening enables the user to activate DHCP or to set the correct IP address:

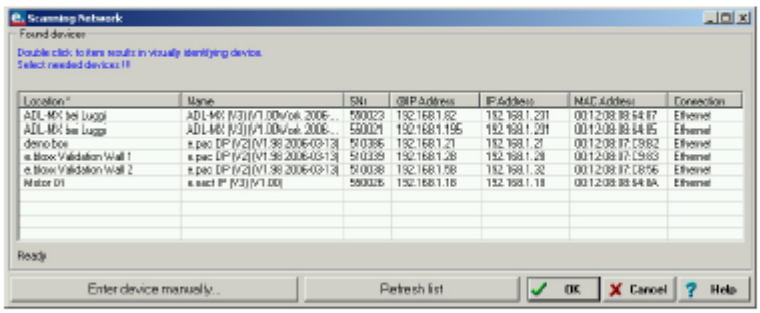


Beside this very comfortable way there is another possibility availble to find an e.reader on the Ethernet. This procedure will be described in the following section:

3.4 Traditional way of finding an e.reader on the Ethernet

Each device on an Ethernet has a unique address. These addresses are used to find and communicate with the corresponding devices. On delivery the address of an e.reader is set to 192.168.1.18 as default, with DHCP set to ON.

After selecting the option field "Open new project and read online system..." in the start-up window the test.commander will search the network for all connected Ethernet devices and show them in a new window.



If the e.reader could not be found there might be one of the following problems:

1. Connection problem: Check that the e.reader is connected correctly to the Ethernet and the power is supplied to the module (LED "RUN/ERROR" is on).
2. The IP address of the e.reader does not fit to the network it is connected to.
3. Wrong DHCP settings: By default the DHCP setting of an e.reader is set to ON. In this case the e.reader will wait for a DHCP server in the network to get an IP address assigned. If there is no DHCP server in the network or the e.reader is directly connected to the PC/laptop via an Ethernet cross-cable, then the DHCP setting must be set to OFF (see chapter 3.4).
4. The IP address of the e.reader conflicts with another device on the network: In this case refer to the next chapter 3.4 to change the IP address.

Select the e.reader you have to work with and click "OK". The test.commander will read the configuration of the e.reader module and it will be displayed on the screen (see 3.5).

Note: By double-clicking on an e.reader the "RUN/ERROR" LED of the e.reader will start blinking fast for a few seconds. This indicates if this is the system you are going to work with.

Setting the Ethernet Address of an e.reader Module via the RS232 interface:

An e.reader can be connected directly to a COM port of a PC/laptop via the serial RS 232 interface cable that is included in the scope of supply. This way it is possible to change the IP address and the DHCP setting of the e.reader.

In the test.commander software select the menu item "Concentrator Terminal..." in the menu "Utilities". The following window will open.

The e.reader, that is connected at the COM port, will be read and the address information of it will be displayed in the window. If no module is found check the interface settings (correct COM-Port) in the "Options" Menu and click on "Read".

The actual settings of the e.reader will be shown in the window. The settings can be changed here. Enter an IP address which is not yet used in the network the e.reader shall be used later on. Also check if the field "use DHCP" is set correctly. If there is a DHCP server in the Ethernet, where the e.reader shall be used, mark this field. Otherwise keep it unmarked. A DHCP server is used to assign an IP address dynamically to the devices in the network.

Note: No leading zeros are to be entered in the IP-address (valid = 192.168.1.18, invalid = 192.168.001.018).

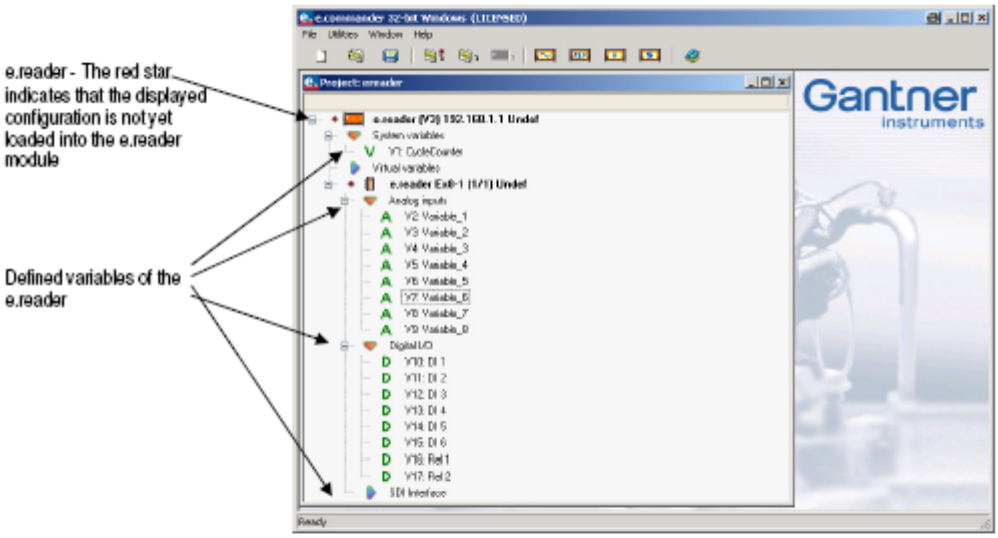
3.5 Read Configuration from an e.reader

To see the configuration of an existing e.reader module add the module to a project. Therefore press the right mouse button in the project window and select "Add Online Concentrators..." from the pop-up menu. The test.commander will search the network and display all e.reader and test controllers on the network in a list as described in the chapter 3.3.

Select the desired e.reader and press "OK".

During the next few seconds the communication between the PC and the e.reader is checked, the configuration of the system is checked to see if the configuration correlates with the hardware, to verify the e.reader, to test the communication aso.

As a positive result of the above mentioned system check the project is displayed in a window with a list of all the defined variables.



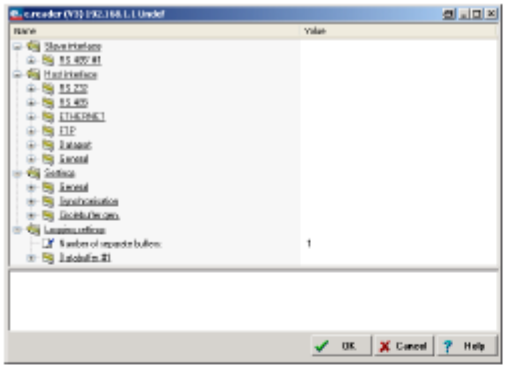
By double-clicking a variable its settings can be changed. See the chapter 5 "Configuration" for more information.

3.6 Edit a Configuration

It will be distinguished between the configuration of e.reader specific settings and the configuration of the variables of an e.reader.

3.6.1 Configuration of e.reader

To configure the settings of an e.reader double-click onto the e.reader module in the project window. A new window like the following one will open.



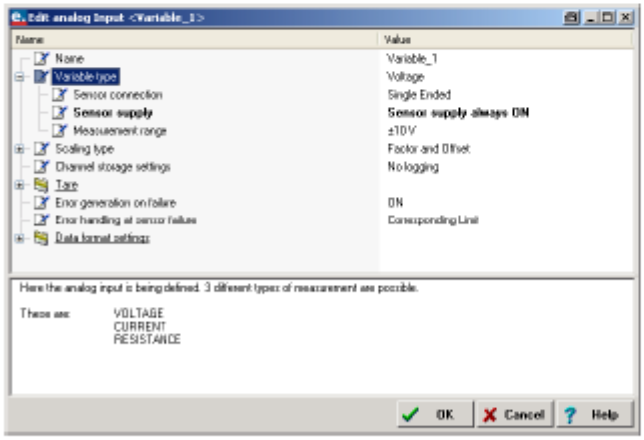
Here all the settings for the e.reader are listed in a tree view. You can change all the settings directly in this window. Beside the interfaces and setting the logging functionalities can be defined as well.

3.6.2 *Configuration of Variables*

With the e.reader four different types of variables (resp. channels) will be displayed. These are the analog variable, the digital variables, the system variables and the virtual variables. Virtual variables e.g. are being used if one variable should be part of different arithmetic variables,..., a system variable e.g. could be cycle counter.

To define all the settings for a variable of an e.reader double-click onto the variable in the project window.

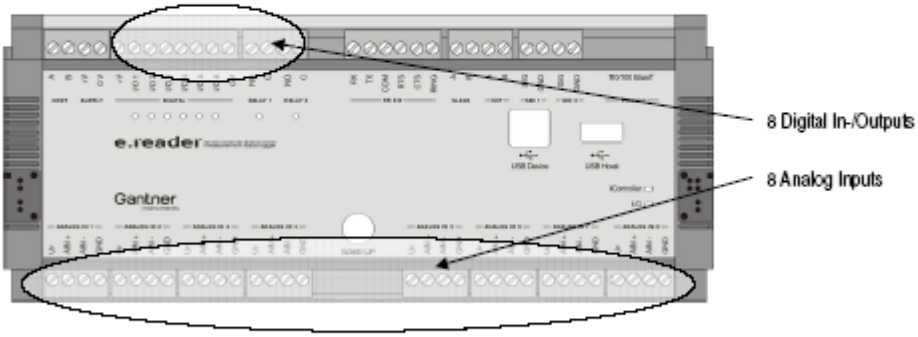
The following configuration window will open:



Here you can define the name and the kind of measurement for the variable and make much more settings like the scaling type, how the logging of the variable shall be handled, the error handling, etc..
In the second window the online help for the corresponding setting will be displayed.

3.7 Sensor Connection

The measurement sensors and output circuits are connected to the e.reader via the corresponding screw terminals. This has to be done according to the configuration of the e.reader, so that the configuration corresponds to the actually connected sensors.

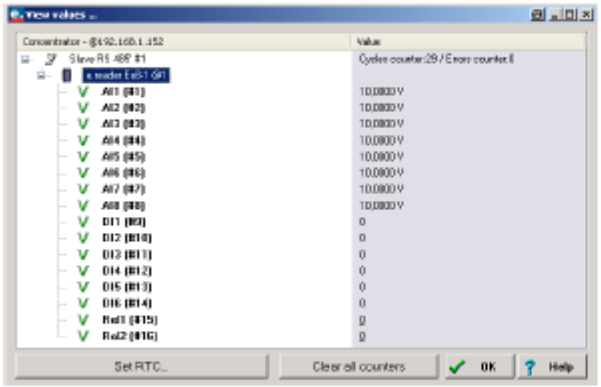


A detailed description how the different types of sensors for the different types of measurement are connected can be found in chapter 4 - "Sensor Connection".

3.8 Display Measurement Values

After configuration and correct sensor connection the actual values of each variable can be displayed and monitored on screen in real time.

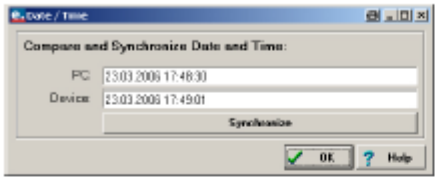
To display the real time values click on the icon "Read Online Values from Concentrator". The test.commander will scan the network and display all connected test controllers in a window. Select the desired e.reader module and press "OK". The e.reader will be read and the defined variables with their actual values will be displayed in a new window.



In the left part of the window all defined variables are listed. On the right side you can see the measurement values of each variable.

With the button "Clear all counters" all counter values of the e.reader can be reset to zero.

Each e.reader has an internal real-time clock for timing information. This clock can deviate after longer time of operation. In order to check the clock press the button "SetRTC". A window will open where you see the time of the e.reader and the local time of your PC/laptop.



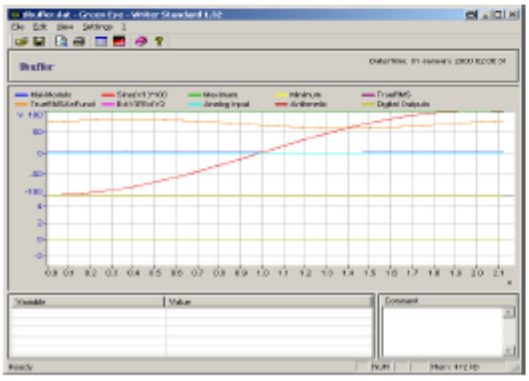
With the button "Synchronize" you can set the clock of the e.reader to the PC/laptop time. Pay attention that this time is set correctly before you set the RTC of the e.reader.

3.9 Read Memory

The e.reader saves all measured values in its internal memory, which is a ring buffer. If some data only have to be logged at a reduced speed this has to be programmed with the test.con software.
In order to read out the buffer of an e.reader click on the icon "Read Online Buffer from Concentrator". The test.commander will scan the network and display all connected test controllers in a window. Select the desired e.reader module and press "OK". The buffer of the e.reader will be read. If you want to store the data locally on the PC/laptop for later evaluated and processing, select "Yes". You will be prompted to enter a file name for the data. If data have to be displayed only, press "No".

Attention: Reading data from the e.reader deletes all measurement data – data can be read only once! Therefore always store data if you need them for further processing!

To display data the test.commander starts the integrated tool "Green Eye" and displays data in a new window.



The licensed version of Green Eye Writer provides helpful features such as difference measurement, zoom function, many scaling and design tools etc. With an unlicensed version of Green Eye it is only possible to view the measurement data.

Note: A detailed description about the evaluation of measurement with Green Eye can be found in the description of the test.commander software.

3.10 Test.con (optional)

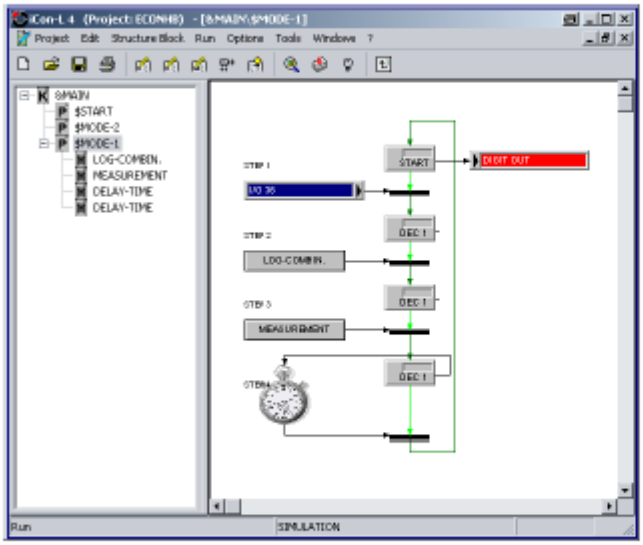
The software test.con is an easy-to-handle graphical programming system to define the functionality of an e.reader module. The programming is done by combining several "function blocks" in certain ways. The whole functionality of these function blocks is defined in the e.reader. The software test.con is just a tool for combining and "wiring" the single blocks.

Programming can be done on a data flow basis (function block language) or by following the control flow (sequences, flow charts). The presentation of graphical blocks follows existing specification and technical languages. Structure blocks permit a hierarchic structure of projects. Linked projects and export mechanisms facilitate the reuse of structure blocks that have been created earlier.

The software test.con combines programming, simulation, testing and running in one tool. Special blocks and additional tools permit online observation of signals and signal tracing as well as run-time measurements.

Detailed information on the test.con software is available in the online help, the test.con introduction manual provides an overview on its possibilities and how to define a project.

The following picture shows a structure for a test cycle.



4. SENSOR CONNECTION

4.1 General

The e.reader has integrated measurement functionality which provides all together eight analog inputs, six digital in-/outputs and 2 relay-outputs for measurement of sensor signals. Depending on the type of sensor, which is connected to the analog inputs, various numbers of terminals have to be used. The configuration of the inputs and outputs is done in the configuration software test.commander according to the requirements of the application. The in-/outputs can be configured independently from each other.

4.2 Analog Input

An analog input collects and processes the signals of voltage, current and resistance sensors. Currently data of standardized and proprietary sensors are stored in the e.reader. The user can input further sensor data. The acquisition of various measuring values with these sensors may be reduced to a few principles of measurement, which are:

- Measurement of Voltage
- Measurement of Current
- Measuring a Resistance

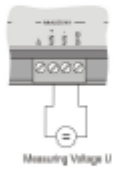
For some of these measurements the e.reader offers several types of measurement. Currents up to 0..20mA may directly be measured by the e.reader. Measuring the voltage drop at an external shunt can carry out measurements of currents over 20 mA. Resistance measurements can be carried out directly.

The analog inputs are protected against excess voltages.

Notice: Overloads of more than ±15 VDC will result in false measurement data at the analog input variable.

4.2.1 Measurement of Voltage

Connection Scheme



Circuit

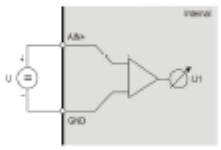
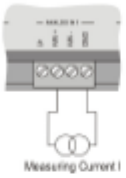


Figure 4.2 - Measurement of Voltage - Single-Ended

With the single-ended type of measurement the voltage to be measured is connected between the terminal "AIN +" and "GND" of an analog input (ANALOG IN 1 .. 8). The measurement voltage may not exceed 10 VDC.

4.2.2 Measurement of Current

Connection Scheme



Circuit

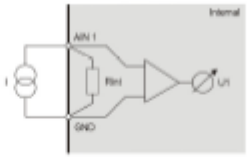
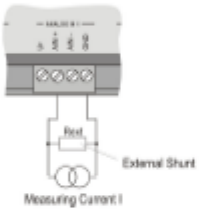


Figure 4.3 - Measurement of Current with Internal Shunt

Connection Scheme



Circuit

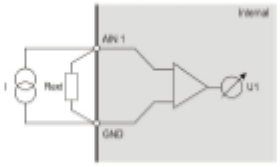


Figure 4.4 - Measurement of Current with External Shunt

For measurements of current with the e.reader the source of electricity is connected to the terminals "AIN +" and "GND" of an analog input (ANALOG IN 1 .. 8). For the measurement, the required load on the current source is regulated by an internal resistor R_{int} with a value of 100 Ω. The maximum power of this shunt is limited to 0.25 W, resulting in a measuring range of up to 20 mA maximum.

If higher currents need to be measured, an external resistor that is connected parallel to the source of current should be used. The power of the external shunt has to be adapted to the source of current to be measured in order to limit the voltage at the analog input to ±10 V. The analog input is configured as voltage input. The voltage has to be divided by R_{ext}.

Notice: The precision of the current measurement with external shunt depends on the accuracy of the resistor that is used.

4.2.3 Measurement a Resistance

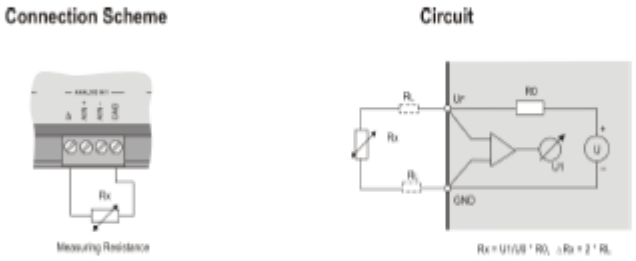


Figure 4.5. - Measurement by a Resistance in 2-Wire Technique

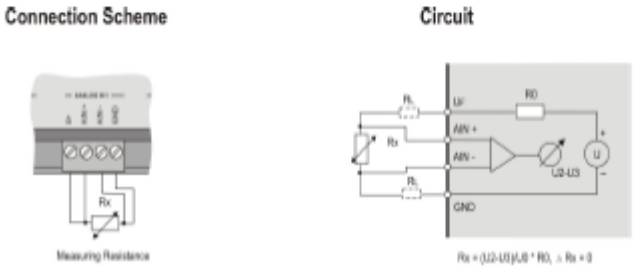


Figure 4.6. - Measurement by a Resistance in 4-Wire Technique

Resistance measuring are carried out by means of measurements of voltages at a current-carrying resistor. In this case the occurring fall of potential is measured via the resistance sensor. The feed current required for the resistance measuring provides the internal supply of the e.reader.

For this purpose the e.reader connects a supply point internally with the analog measurement input via a reference resistor R_o. The fall of potential U_o via the resistor R_o is required as a reference for further signal processing by the e.reader. The value of resistance of the sensor can be calculated from the input signals U_i as a multiple of the reference resistor R_o. The measuring range amounts to between 0 and 20 kΩ.
Notice: The e.reader supports resistance measuring in 2- and 4-wire technique. With resistance measuring in 2-wire technique the supply lines cause an additional fall of potential, thus distorting the measuring result and influencing the measuring accuracy. Therefore it is necessary to pay attention especially with resistance measuring

4.3 Digital Inputs / Outputs

4.3.1 Digital Inputs

The digital inputs of the e.reader can be used for trigger functionality, for status signals, counters, interval counter and frequency measurements. The inputs have an excess voltage protection (transil diodes), which comes into action at approx. 33 V. The maximum permissible input voltage amounts to 30 V. Input voltages between 3.5 VDC and 30 VDC are interpreted as logic HIGH ("1"), input voltages lower than 1.0 V as logic LOW ("0"). The maximum fan-in current amounts to 1.5 mA.

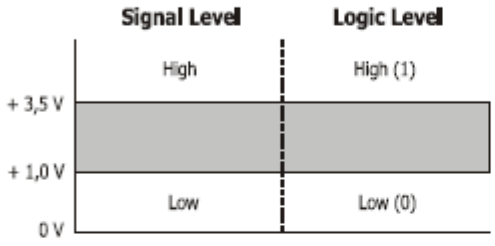


Figure 4.1. - Definition of Signal Levels and Logic Levels

4.3.2 Digital Outputs

The Digital Outputs can be used for status signals and alarm variables.

The alarm variables are configured with the test.commander software and can be used to provide signals e.g. in case of a sensor break.

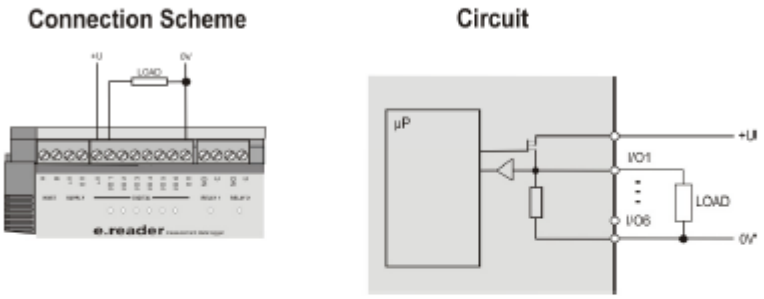


Figure 4.8 - Digital Inputs/Outputs of the e.reader

Figure 4.10 - Signal Diagram of Digital Inputs/Outputs

The digital input is set (switch closed) as long as the applied signal voltage remains under the threshold value of 1 V.
The digital information can be scanned as 1/0 information via bus.

The status of the digital output can be scanned as 1/0 information via bus.

4.4 Relay Output

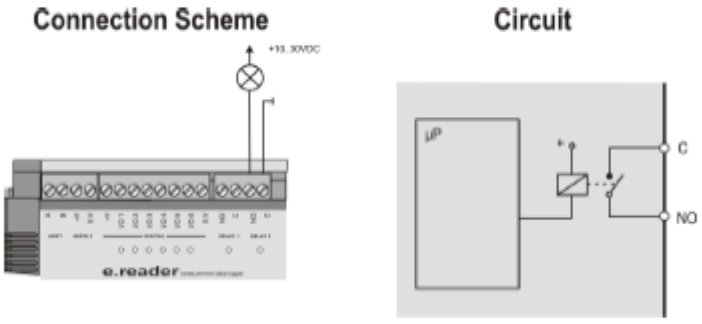


Figure 4.7. - Analog Signal Output

5. CONFIGURATION

To configure an e.reader and read the measured and logged values from the e.reader the Configuration Software test.commander is used. This chapter will give you an overview on how to configure an e.reader module. A more detailed description about the test.commander software can be found in the test.commander manual.

The configuration can be divided into the following processes:

- Definition of module-specific settings for the interfaces, logging, etc.
- Configuration of the measurement variables
- Definition of the functionality of the e.reader by using the test.con software

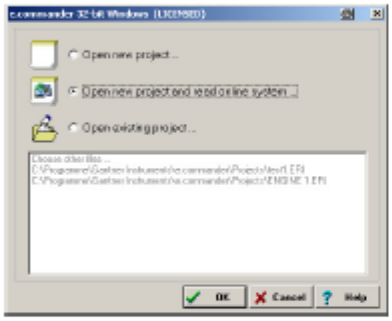
There are two possibilities for configuring an e.reader. With the first possibility the settings and configuration of an e.reader will be read from the e.reader, adjusted and transmitted back to the e.reader.
Another possibility is to define the configuration locally on the PC/laptop without the necessity that the e.reader is already connected or present in the network. The configuration can be saved and loaded into the e.reader at any time.

The entire configuration of an e.reader or a whole system of e.readers is saved in a project. Projects can be saved and loaded for further processing.

The standard way of configuring an e.reader module is to connect the e.reader to the Ethernet first. Then the e.reader will be added to a project, which will read out the actual configuration of the e.reader and the configuration is being display on the screen. Now it is possible to reconfigure the module and finally update the e.reader with the new configuration.

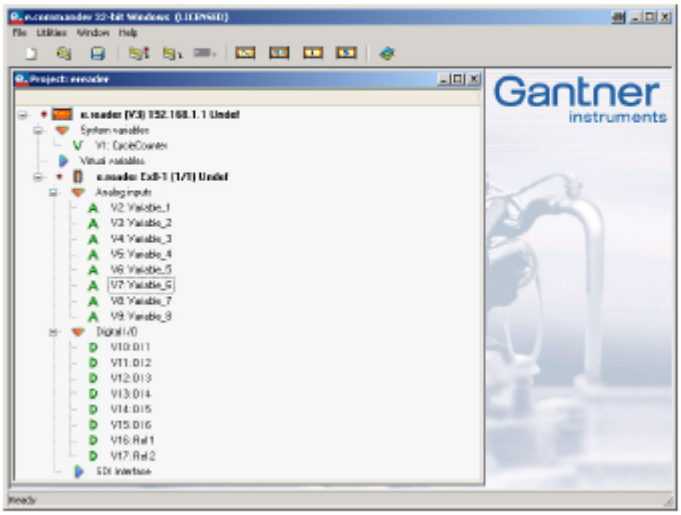
5.1 Open a Project

Start the test.commander software. First you have to open a project. Therefore select one of the three options in the window that will be opened after start-up of test.commander.



- With "Open new project..." you can open a new, empty project.
- Selecting "Open new project and read online system..." will open a window where test.commander displays all e.readers found on the Ethernet. You can select the e.reader you want to work with and press "OK". This will open a new project and add the selected e.reader with its actual configuration.
- With the option "Open existing project..." you can open a previously saved project.

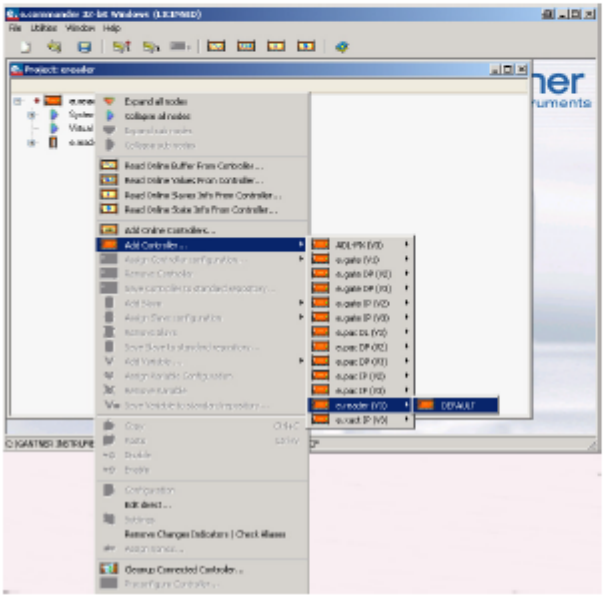
You can open several projects at the same time as well. Each project will be displayed in a separate project window.



Below each e.reader in a project the variable settings of the in-/outputs of the e.reader and the configuration of the slave modules e.bloxx, which are connected to the e.reader, will be shown in list form. It is possible to add new e.readers and slave modules and to define the variable and device settings.

5.2 Adding an e.reader to a Project

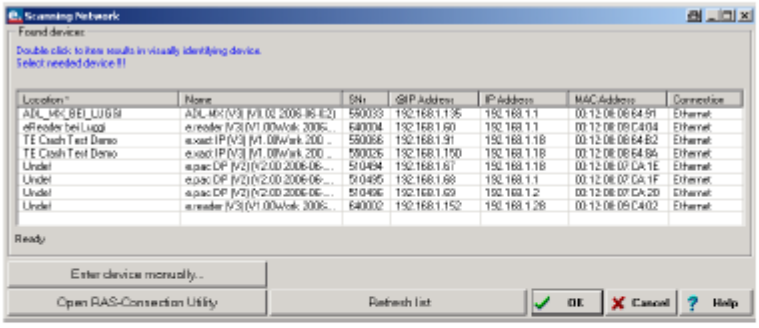
To add an e.reader to a project click with the right mouse button below the last line of the project. A context menu appears.



Now you have the possibility to add an e.reader which is already working and present at the Ethernet or to add a new e.reader module which is not available yet but you want to make its configuration already.

- In case you want to add a new e.reader which is not yet available on the Ethernet select the menu item "Add Concentrator..." and select the corresponding type of test controller you want to add (e.g. "e.reader").

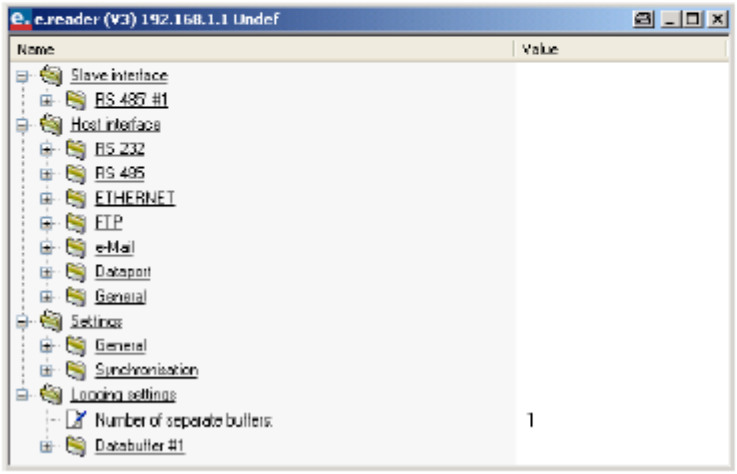
- If you want to add an e.reader which already exists on the Ethernet select the menu item "Add Online Concentrators...". The test.commander will then search the network and display all test controllers on the network in a list.



With "Enter device manually..." you can enter the IP-address of an e.reader directly. With "Refresh list" the network will be scanned again and the list of modules will be updated. Double clicking one of the e.reader in the list makes the LEDs of the e.reader blinking for a few seconds. This indicates the position of the e.reader you are going to work with. Mark the desired e.reader and press "OK".
After one of the two possibilities just mentioned the project window will be shown again and the e.reader will be added to the configuration list in that window.

5.3 General controller / e.reader settings

To change the settings of an e.reader double-click the mouse onto the e.reader or click on it with the right mouse button and select "settings" from the context menu. The window to define the settings will be opened.



The settings are arranged in a group for the "Slave Interfaces" (connection of e.bloxx sensor modules), the "Host Interface", the e.reader specific settings and the logging settings.

Following is a short overview of the setting possibilities. By clicking with the mouse on a setting a help text for the selected setting will be shown in the lower area of the window.

5.3.1 Slave Interface

The e.reader provides one slave interfaces for the connection of additional e.bloxx modules via the screw terminals A_{Slave} and B_{Slave} of the e.reader. See chapter 5.5 for information on how to add an e.bloxx to the slave interface. In the following the slave interface is indicated as UART1. The settings like baud rate and delay time are selectable. The protocol is fixed to Localbus, 8e1. This setup will overwrite the setups in the module to ensure a communication.

5.3.2 Host Interface

To connect the e.reader to a host there are several interfaces available. The most important interface is the Ethernet.

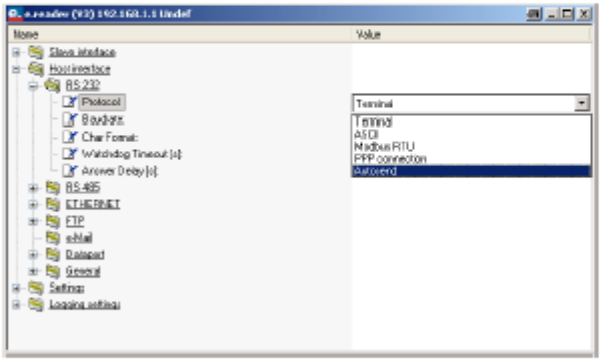
By using DHCP it is necessary that the Static IP Address and the Default Gateway Address fit together. The actual dynamic IP address can be displayed in the *General* selection.

Within the function *FTP* a password protection can be activated. It is necessary to work without or with all 3 passwords. In case of forgotten passwords please contact Gantner Instruments at office@gantner-instruments.com to unlock the password protection.

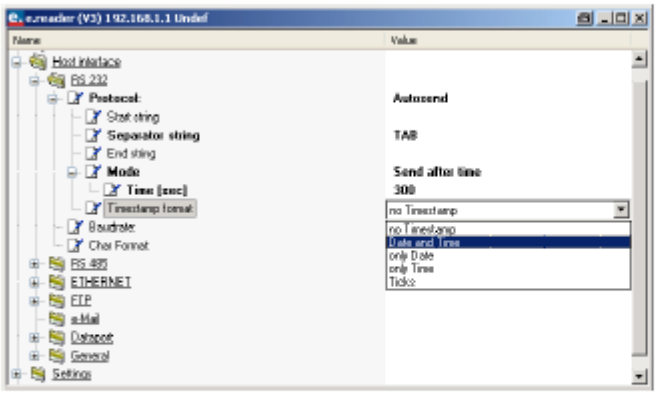
Another very interesting way of communication is via e-Mail. This handling is being described later in this manual.

Configuration of RS232 / RS 485:

To transfer data from the e.reader several interfaces are available. If the Ethernet interface will not be used, the RS 232 or RS 485 interfaces are available as well.
To configure these interfaces select the e.reader with the right mouse button in the test.commander and choose "Settings". In the section "Host interfaces" the RS 232 and/or RS 485 interfaces can be selected for configuration. Depending on the application the required protocol has to be selected:



Terminal is being selected to configure the e.reader (e.g. IP-address), for modem communication you e.g. select the PPP connection or if data have to be sent automatically you will select "Autosend".



With Autosend the Start- and End-String has to be defined as well as the separator string. Now different modes are available such as "Do not send", "Send all variables", "Send after time" and "Send only buffer". Choosing "Send after time" requires a time in secondes when data have to be sent, e.g. every 300 seconds.
Even for the timestamp format there are different possibilities such as "no Timestamp", "Date and Time", "only Date", "Only Time" and "Ticks".

5.3.3 e.reader Settings

The e.reader settings are separated into 6 sections:

In the section "General" a designation can be assigned to the e.reader. With this name it will be indicated when searching for test controllers on the Ethernet and also in the project window. As default setting the "ePac Functionality" is deactivated. If you require ePac functionality you will need the additional software package test.con. For details on this software tool please refer to chapter 6.
The "Power down mode" is deactivated as a default as well. In this case the e.reader is running all the time, the energy saving functionality is being switched off.

Note: When changing a setting, the setting will be marked in bold text. When closing the setting window the changes will be written to the project file of your PC/laptop. To activate the new configuration in the e.reader the configuration file has to be sent to the e.reader module as well.

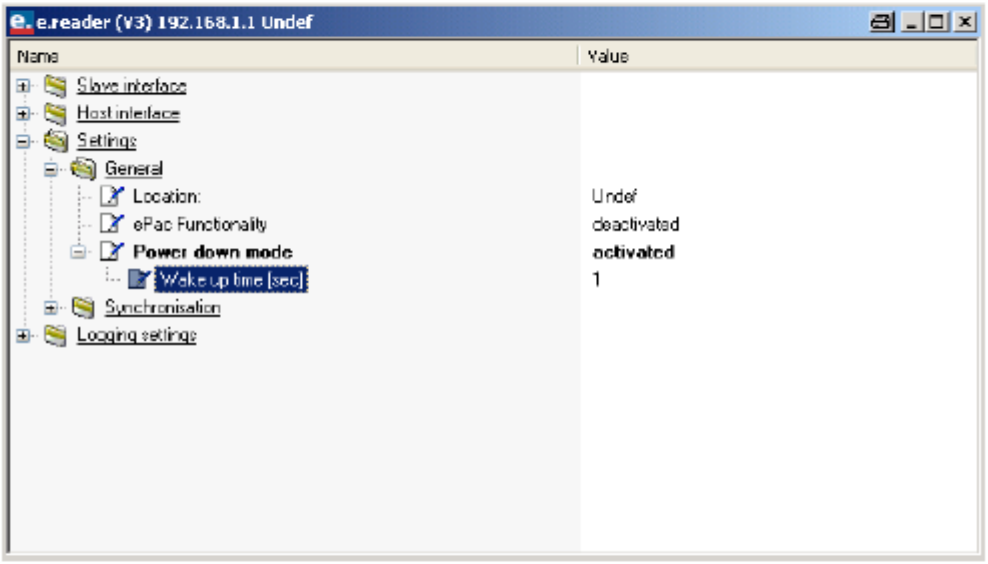
The circlebuffer in the e.reader stores measured values for later read-out and evaluation. It is possible to select the number of circle buffers (1 or 2) and set the number of variables to be stored (simultaneous or sequential storing), the buffer size, sample divider (e.g. a fast and a slow buffer), trigger mode and trigger position (pre-trigger).

5.3.4 *Low-Power-Functionality- Power down mode*

The e.reader provides 2 different modes, the standard mode and the low-power-mode.

The standard mode is being used if the logger is being connected to a standard power supply, if no reduction in power consumption is required.

With some applications (e.g. power supply via a solar pannel or a battery) a very low power consumption will be required. Therefore the e.reader provides the "Low-Power-Functionality". This functionality has to be defined in the e.reader settings:



The following section describes the "Low-Power-Functionality" in more detail:

If the "Power down mode" is activated, the cpu of the e.reader will be "switched off". Only the data acquisition part will be active during that time. This will result in the following handling of acquired data:

- no arithmetic calculations will be done
- do digital I/O will be active during that time if they depend on e.g. a threshold function.

After the defined "Wake up time" the cpu will wake up and handle the acquired data, e.g. do mathematical calculations, store data to the flash memory, switch digital I/Os.

In general there are 3 possibilities to wake up the cpu board:

- by the defined "Wake up time"
- if you connect to the e.reader e.g. via a modem
- if you press the wake up button on the front part of the e.reader

Remark: If a "Wake up time" of less than 20 seconds is being defined, the low power mode makes no sense, you will not save any energy. It is strongly recommended to activate the Low-Power-Functionality only if the application requires a very low power consumption!

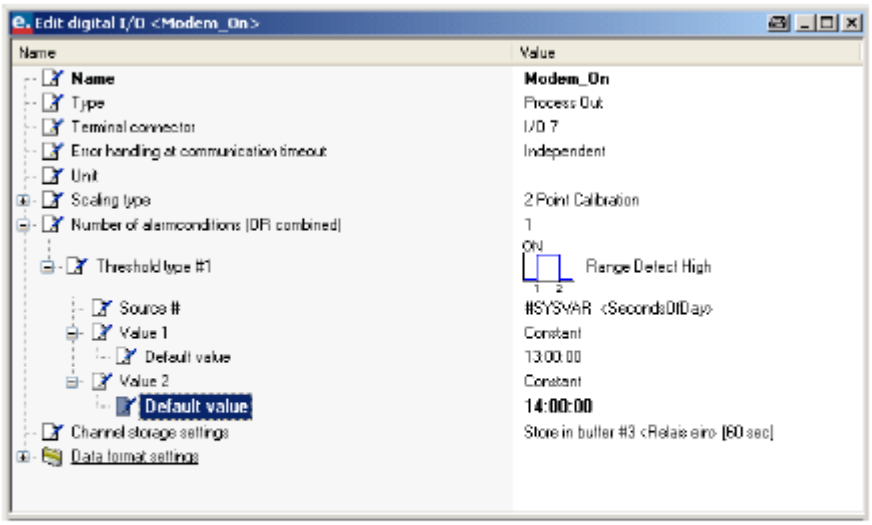
The following table provides an idea on the power consumption depending on the sample rate:

Sample rate	Power Consumption	Remark
< 20 seconds	approx. 4 W	the "Low-Power-Functionality" makes no sense as mentioned above
1 minute	approx. 200mW	
10 minutes	approx. 100mW	

5.3.5 Low-Power-Functionality- Modem Communication

As there is no communication with the e.reader possible in the low power functionality a special wake-up handling is required.
For a standard communication via modem the connectors RX, TX and COM are required. With the sleep mode the additional RING connection has to be used. Calling up the connected modem this terminal makes the e.reader recover from the sleep mode, the cpu is being switched on. Now communication with the e.reader is possible and all functionalty is available.

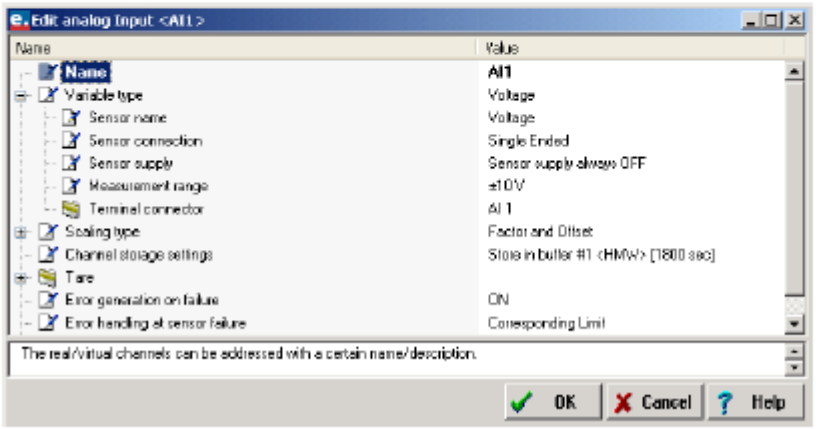
In some cases it will be necessary to switch off the modem as well to save power. In this state it is not possible to communicate with the e.reader via a modem, even if the RING terminal is connected. With the configuration software test.commander you can define e.g. a certain time-window when the modem should be switched on using a relay output. The following picture shows a possible configuration of the relay output:



In this example the relay output is switched on between 1 and 2 p.m. As a source the system variable "Seconds of Day" will be used and the values are being defined as "Constant" and the required time. During that time the modem is power supplied and the e.reader can be waked up by using the RING terminal.

5.4 Configuration of the Variables of an e.reader

To change the configuration of the measurement variables of the e.reader double-click on the variable in the project window or click the variable with the right mouse button and select "Configuration". A window like the example below will be opened, where you can see the actual settings of the variable. The information in the window depends on the type of variable (analog, digital, input, output).



The variable name (has to be unique for every variable in case you will work with the test.con software), the variable type, scaling and range settings, the tare functionality, the error handling and data format settings for the selected variable are indicated here. Click on a "Value" field if you want to change one of the settings.

Note: All the settings having been changed and not yet written into the e.reader will be marked in bold text.

When you confirm with "OK" all changes (highlighted in bold font) will be saved and written to the e.reader. The project window will be shown again.

Note: Pay attention that the physical sensor connection coresponds to the defined configuration.

Following is a short overview of the setting possibilities. By clicking with the mouse on a setting a help text for the selected setting will be shown in the lower area of the window.

5.4.1 Variable Type

Here you can define the type of measurement for the variable. Depending on the variable type (analog, digital, input, output) different types of measurement are available. The field "sensor connection" lists all measurement functions of the selected variable type.

If there is a sensor connected which requires a supply voltage from the e.reader you can select if the supply shall be on always, off or optimized (on if required).

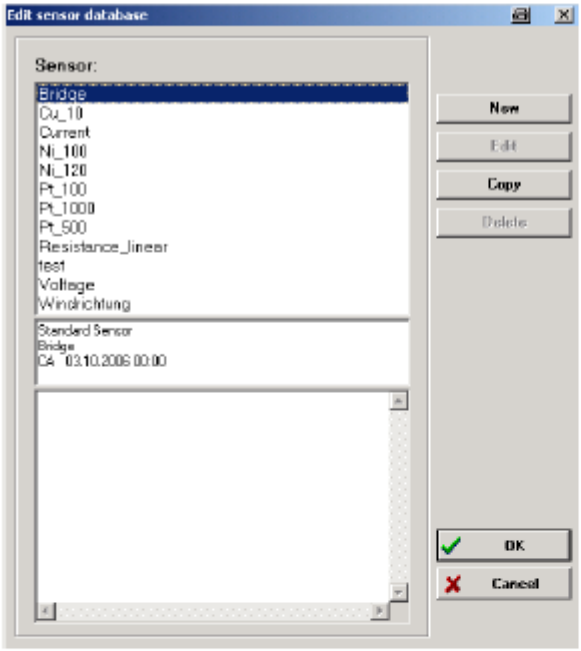
The range of measurement can also be defined. Adjust this to the predictable maximum and minimum measurement value.

5.4.2 Sensor Name

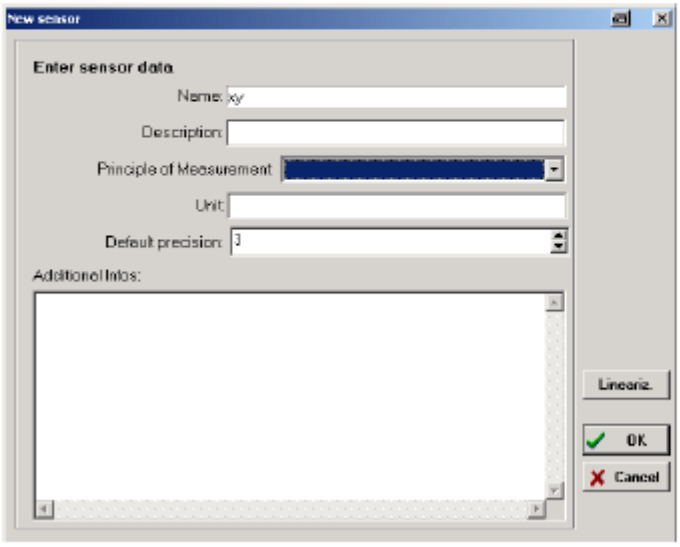
For the different variable types several predefined sensors are available, e.g. Pt100, Pt500,... Depending on the variable type the available sensor name will be displayed and can be selected. In case a customized sensor has to be used, this sensor has to be added to the sensor data base. The following description shows how to add a customized sensor

From the menu "Extras" select "Edit senor database":

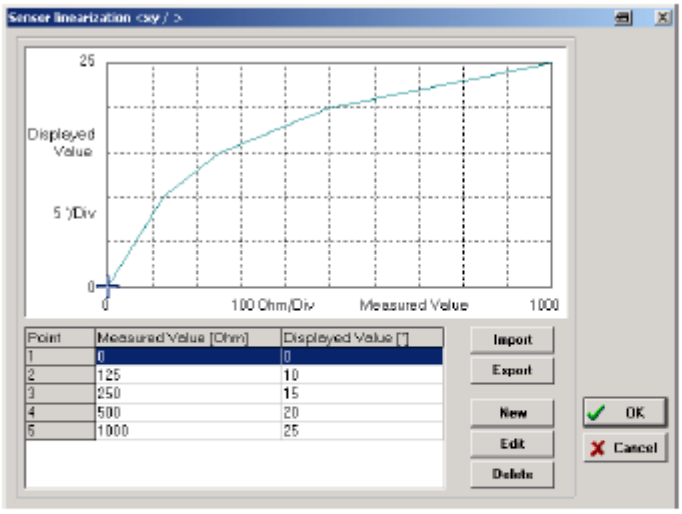
Now a new sensor can be defined. In case a similar sensor is available already you can copy this sensor and do the changes. Otherwise a new sensor can be defined:



Here the details of the new sensor will be added. With the e.reader "Voltage", "Current" and "Resistance" are available with the "Principle of Measurement" – even if some more principles could be selected.



After all the required settings are done, the linearization curve of the sensor has to be defined. Therefore please click the button "Lineariz.":



Here the vales can be added or an available data list can be imported.

Afterwards please confirm all dialogs and the new sensor will be available to be selected via "senor name".

5.4.3 Scaling Type

It is possible to transform the measured value of a variable into another unit, which is called scaling. First select the type of scaling with the list field behind "Scaling Type". With "Factor and Offset" you can enter the factor (multiplication of the measured value) and offset (shifting the conversion curve) for the calculation of the measurement value. With "2 Point Calibration" enter the values of 2 measurement points one time in the measured unit and one time in the conversion unit. For example to calculate the temperature from the measured resistance given by a PT 100 temperature sensor enter two resistance values in Ohm and the corresponding temperature values in °C or °F.

5.4.4 Error Handling

It can be selected if the I/O LED on the front of an e.reader shall be switched on red in case of an error (measurement range exceeded, communication interruption etc).

Also it is possible to define how the e.reader shall react on a sensor failure (e.g. line break). So in case of a sensor failure the value of the variable can be set either to a limit value, stay on the last measured value or set to a certain default value.

5.4.5 Data Format Settings

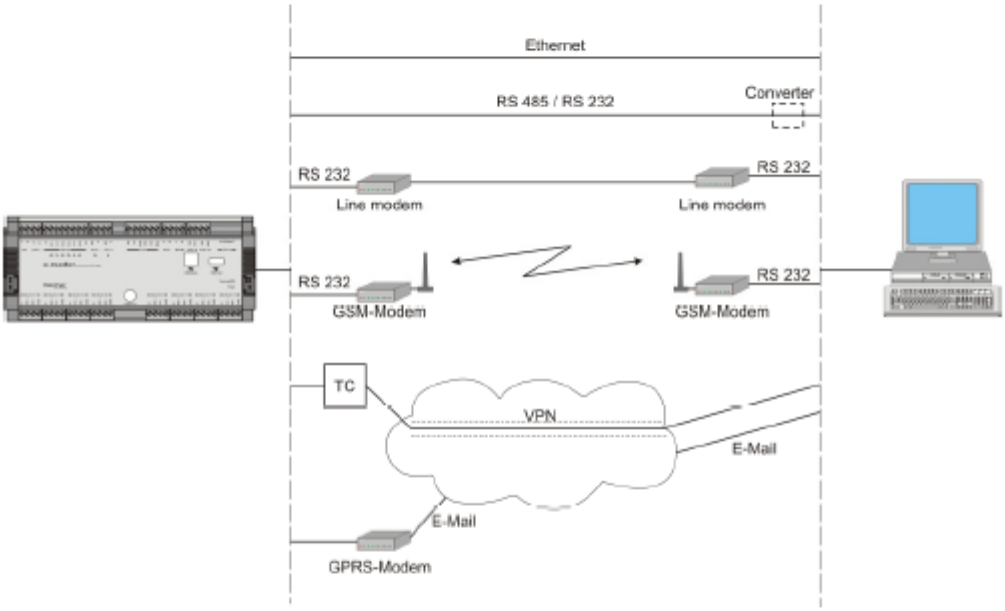
With these fields you can define the data format in which the measured values shall be transmitted via bus. Select the format of the variable (integer, floating real value, boolean, etc.), the field length and the precision. Precision indicates the number of digits after the comma and the field length is the total number of digits before and after the comma.

Data Direction defines if the value is an input or output value but it is possible to set it as input/output as well. Defined as empty the value of this variable will not be transferred, the variable is deactivated.

5.5 Configuration of Communication

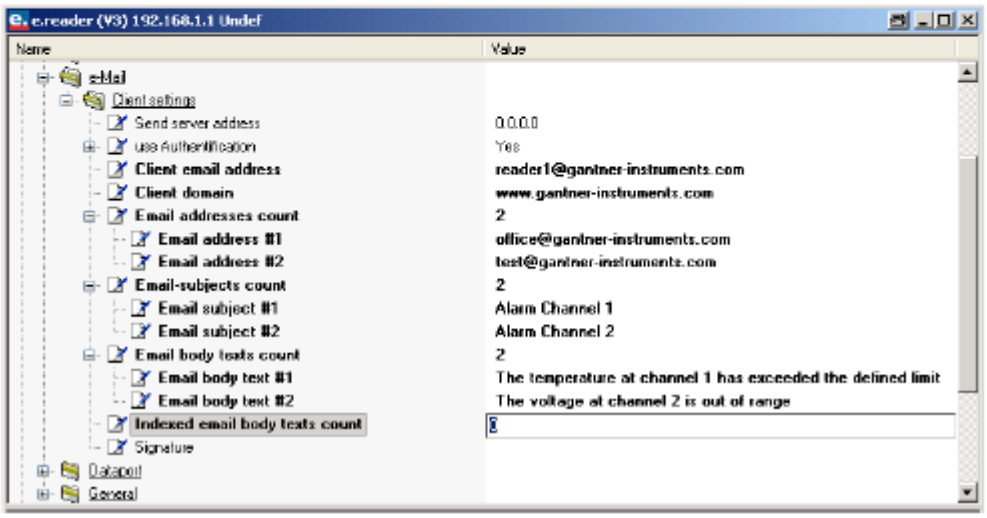
The e.reader provides serveral possibilities for communication. The general setting have to be defined in the e.reader settings, additional information have to be added in the arithmetic channels of the e.reader which are available via the virtual variables. Detailed information on how to configure the e-mail communication is part of the test.commander online help.

The following picture provides an overview on different ways of communication:



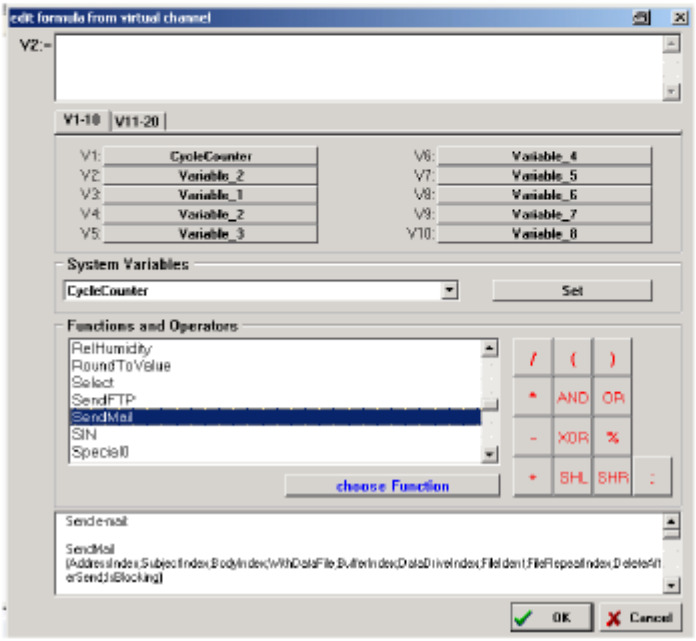
5.5.1 E-mail

First of all, the required information (messages, e-mail-addresses,...) have to be defined within the e.reader settings. Click the e.reader in the system tree with the right mouse button, select "Settings", "Host Interfaces" and "e-Mail". There you have to add all required information. The following picture shows you an example:



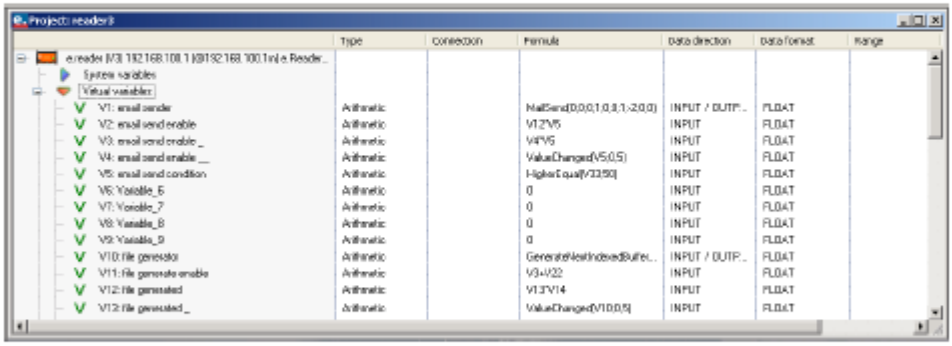
With the e.reader up to 10 different e-mail addresses, e-mail subjects and e-mail body texts can be defined. The general information has to be defined in here, the details/selections have to be defined with the arithmetic variables in the virtual variables of the e.reader.
The e.reader provides the possibility to send e-mails which may contain data from its memory to a server automatically.

In a next step you have to define a virtual variable in the e.reader. A double click on this variable opens the edit window – select "Formula". To create the e-mail you have to select the "SendMail" functionality:



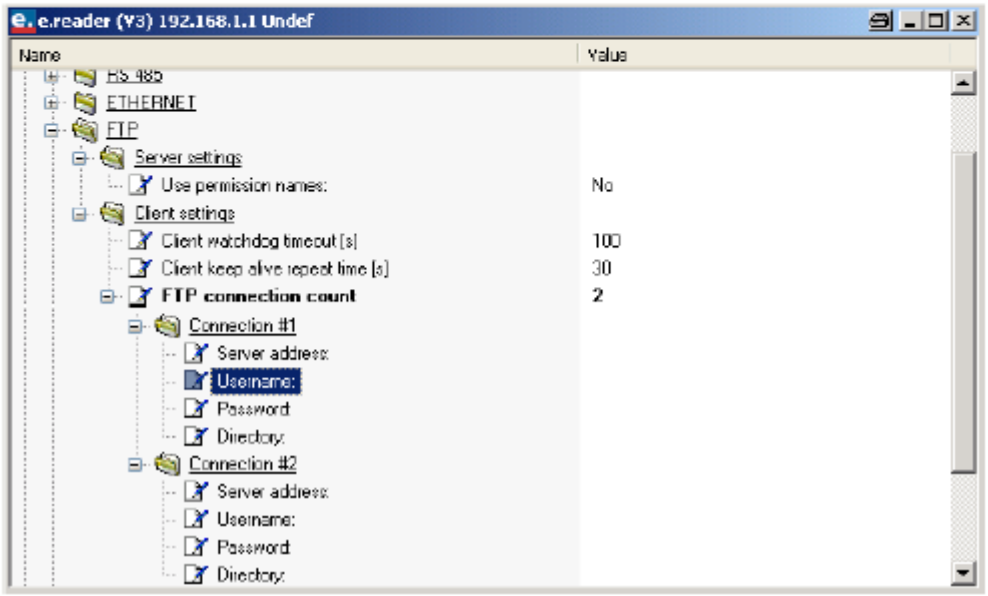
The online help shows all required parameter for the e-mail functionality and its handling and will not be part of this description.

The following example shows a way on how to configure the e-mail functionality. This configuration is built up in a way that an e-mail - including a data file – is being sent as soon as a certain measurement value exceeds a threshold value.



5.5.2 FTP - Client

The handling of the FTP communication is very similar to the e-mail communication. The settings are done in the e.reader settings:



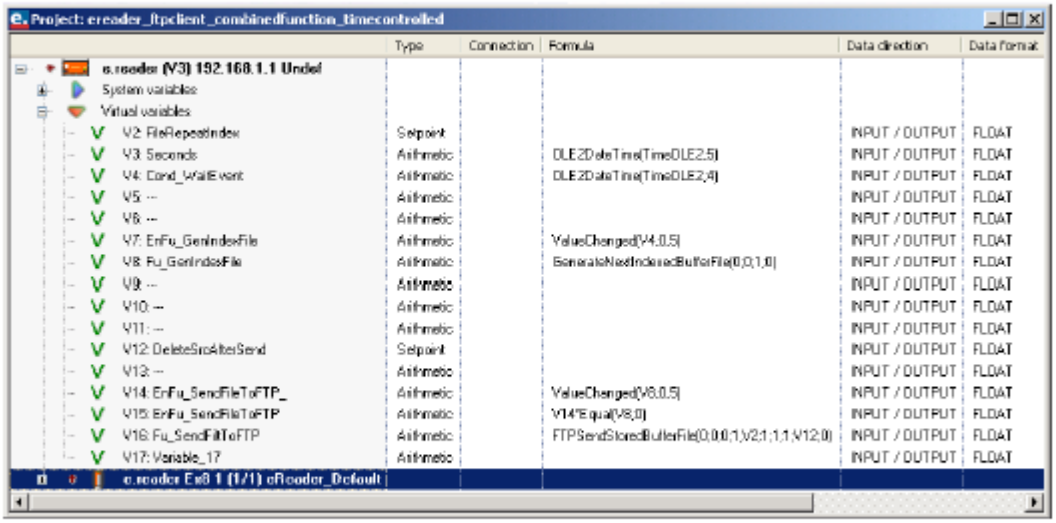
Up to 10 FTP connections can be defined. For each connection you have to define the required information. The handling itself will be done with the virtual variables as well – the formula editor will provide the required handling.

This FTP communication can be used to send data to a server automatically. These data will be stored to the server into a certain directory – and there they are available for further handling.

Attention: Please keep in mind to open an FTP connection before sending data to a FTP server and close it afterwards. The test.commander provides the function FTPSendStoredBufferFile(...) which covers the whole functionality.

To program such an FTP transfer this has to be programmed via the virtual variables of the e.reader. It is recommended to use some variables as spare parts as the parameter are fixed and do not change automatically.

The following picture shows an example on how to configure such a FTP transfer, some comments will be added:



Attention: This design of the window is available with test.commander version 1.4.2 or higher

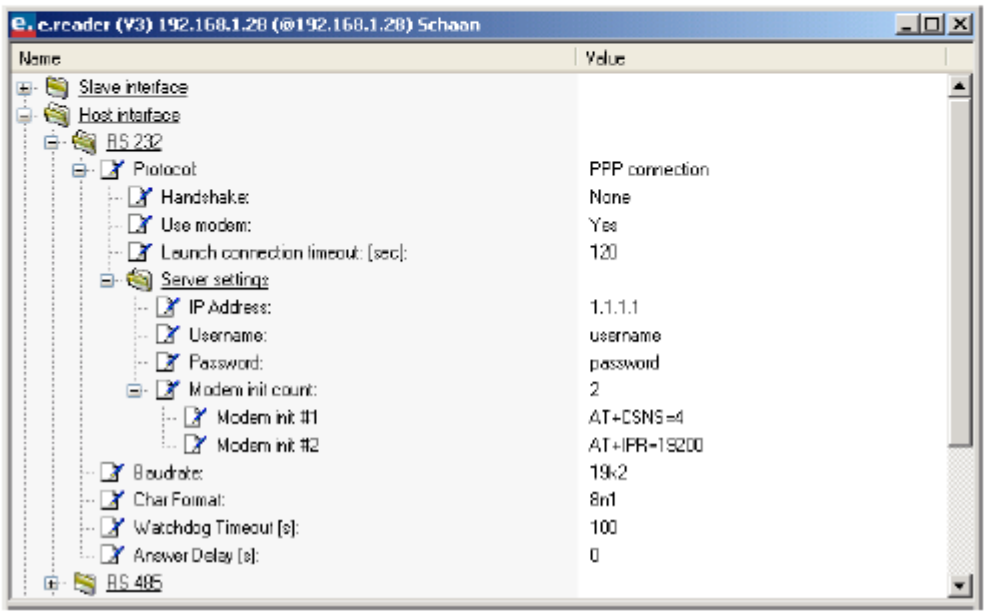
Here is a short description of the FTP-programming – sending a new file after a certain period of time:

V2: The FileRepeatIndex defines which files has to be sent to. E.g. "-1" will send the oldest file, "-2" will send the latest file
V3: this function calculates the seconds of the day.
V4: this function defines the condition for sending a file – in that case every 60 seconds.
V7: here it is defined that an indexed file has to be created as soon as V4 changes from one minute to the next one.
V8: Here the new data file is being generated.
V12: This defines if the source file has to be deleted or not. It is not necessary to do this via a setpoint, it can be fixed as well.
V14: This variable enables the function to send the file in case a new file had been generated
V15: this is the second part of enabling to send the file
V16: this last command will now send the data file.

All parameters mentioned are described in the online help of the test.commander software.

5.5.3 PPP – Connection for modem communication

The e.reader is capable of communicating via PPP. All the settings have to be done with the e.reader settings as shown in the following picture:

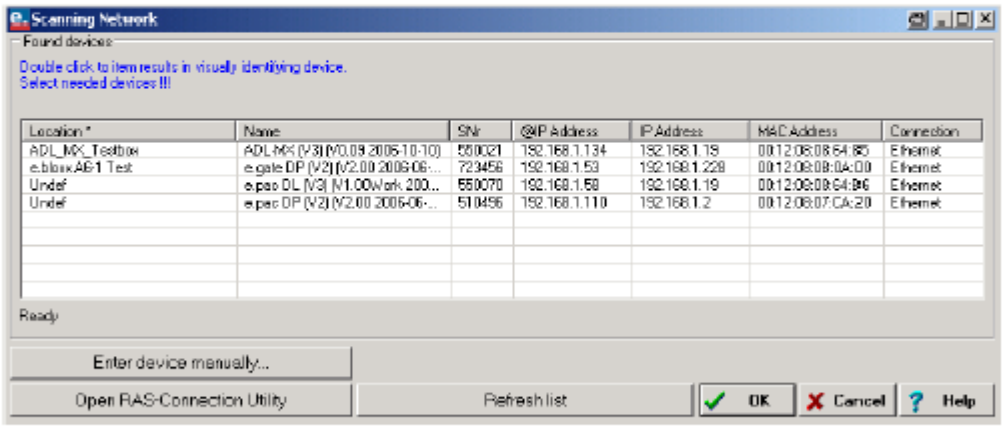


In the section "Server settings" the settings of the e.reader have to be defined. The "Launch connection timeout" will be required e.g. to synchronize the modem for communication.
Depending on the modem type being used different init strings might be necessary. It might be necessary to add one string by another (up to 10 separate strings will be possible), some modems provide the possibility to add several strings into one single command. The example above e.g. requires the init string "AT+CSNS=4" to switch to the data mode in case there is no identification with the communication request. The second init string defines the Baud rate for communication.
The Watchdog Timeout is recommended to be at least 100 seconds. If you decide for a very short time it might happen that the connection will close during the synchronization time.

Attention: It is required to have a PPP connection available with the PC as well!

The following part describes how to communicate via modem and the PPP communication.

First of all a data communication has to be defined on the PC (the same way as it has to be done if a modem has to be connected (e.g. iPassConnect). Afterwards the test.commander has to perform a network scan (as described in chapter 3.4) which will show all known modules in the network.



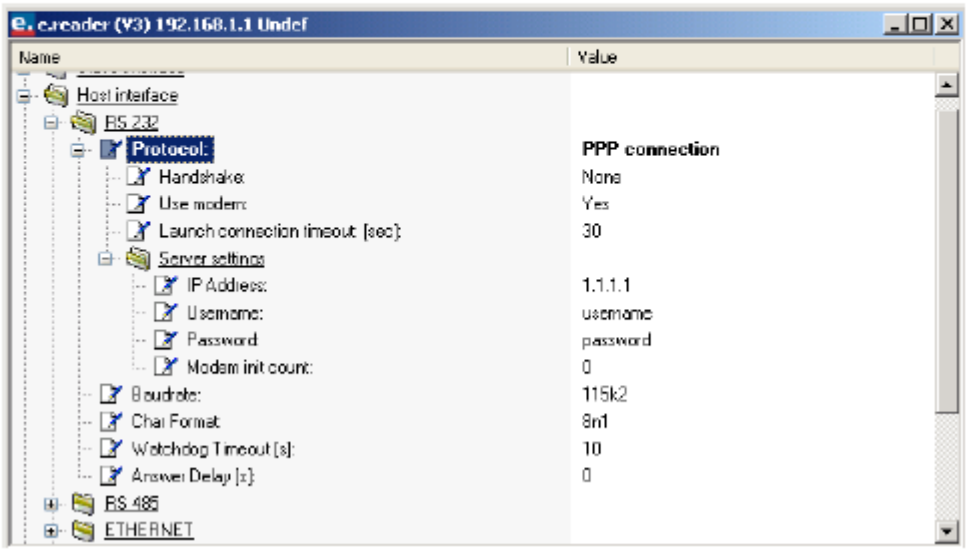
For the PPP connection click the button "Open RAS-Connection Utility":

In the available connections all on the PC defined connections will be displayed. Select the connection you require and the number will be displayed.

If all required information is available click "Connect" to get access to the e.reader.

Details on Username and Password will follow in the next section.

To get access to the e.reader via the modem Username and Password have to be defined – this information has to be equal to the information being defined with the e.reader settings. The following picture shows the required part:

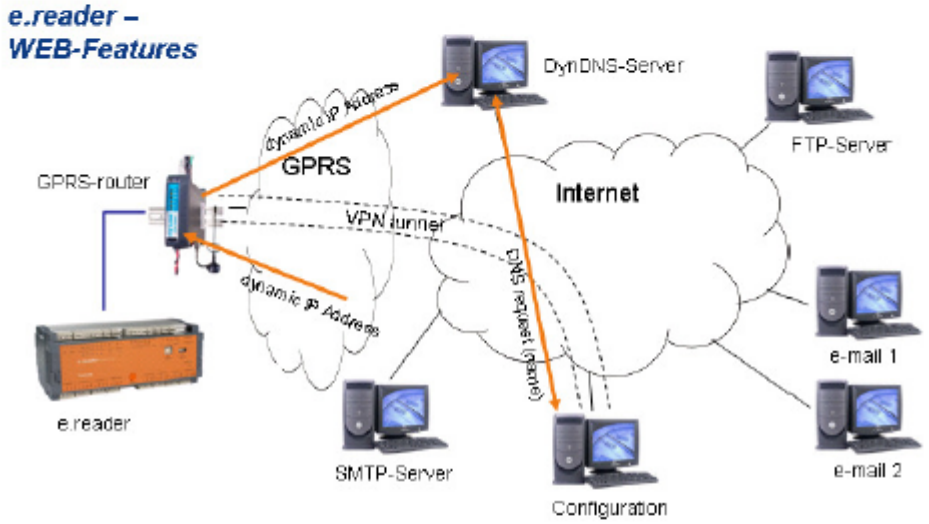


The IP address is the address of the e.reader the connection has to be established with, Modem init count depends on the type of modem being used.

After successful connection the handling is the same as with an online communication.

5.5.4 Truecon – modem with firewall

A communication via the Truecon modem will establish a VPN tunnel. The communication itself is e.g. via GPRS.
The following picture gives an idea on the functionality when establishing an VPN tunnel. In this example a GPRS-router is being used – available at www.hyline.de. This router mentioned is one of several possibilities which could be used. We recommend this router as this one had been tested by Gantner Instruments and it works fine. As the settings depend on the network provider please contact Hyline for technical details – it is also recommended to purchase this router at Hyline directly.



5.5.5 (A)DSL modem

To have access to a remote e.reader it is possible to connect it to an (A)DSL modem via a wired router.
For such a connection there are several requirements for the wired router:

1. The e.reader requires a wired connection to the router
2. Dial in possibility via PPTP (mainly Austria) and/or via PPPoE (mainly Germany) – dependent on the provider.
3. Possibility to enable a constant connection (e.g. after a reboot, resp. break down of the connection an automated dial in is required).
4. Setup of a DMZ computer in the private network (a computer being visible from outside within a secured network) and/or support of port forwarding with freely definable ports (the router should be able to answer on a WAN-ping).
5. DynDNS-support to have access to a domain name instead of an IP-address (which can be different with each new dial in)

5.5.6 USB- Data Transfer

Logged data can be read out e.g. with an USB-stick. The following part will describe how to handle the e.reader with an USB stick:

1. In case the e.reader is in low power mode, the "Wake-Up" button has to be pressed. Now the lower LED is shining green
2. Now the USB stick can be plugged in – the LED starts blinking very fast while recognising the stick and writing data to it
3. As soon as the LED is shining green again (continuously) the USB stick can be plugged off.
4. Data can be visualized with GreenEye directly which is part of the test.commander software. Afterwards data can be stored e.g. as a csv-file.

With an USB stick just the work file (max. 1MB) can be downloaded from the e.reader. When reaching the 1MB limit, internally the oldest 20% of the work file is being cut off and data are being stored in a new file. These files can be downloaded via an FTP connection only.

For applications differing from this basic handling the test.con software will be a very useful tool.

5.6 Data File Management

With the e.reader up to 4 different data buffer can be defined. They will run independent from each other, the logging interval can be different. To identify the different files a special index is being defined:

The following picture shows some files available on the e.reader:

[..]		<DIR>	00.00.1980 00:00 ---
ttactual	ste	612	10.10.2007 00:01 -444
ttsummary	ste	4.290	10.10.2007 00:01 -444
$onlasc	dat	622	10.10.2007 00:00 -444
$onlbin	dat	63	10.10.2007 00:00 -444
$onldes	dat	704	10.10.2007 00:00 -444
^b0_0000	dat	608	02.10.2007 15:30 -666
^b0_0001	dat	4.328	04.10.2007 17:00 -666
^b0_0002	dat	8.088	08.10.2007 17:30 -666
^b0_work	dat	4.048	10.10.2007 15:30 -666
^b1_0000	dat	256	02.10.2007 15:30 -666
^b1_0001	dat	2.504	04.10.2007 17:00 -666
^b1_0002	dat	4.760	08.10.2007 17:40 -666
^b1_work	dat	2.328	10.10.2007 15:30 -666
^b2_0000	dat	893	02.10.2007 15:30 -666
^b2_0001	dat	14.978	04.10.2007 17:00 -666
^b2_0004	dat	133	05.10.2007 10:04 -666
^b2_0005	dat	153	05.10.2007 10:09 -666
^b2_0006	dat	24.013	08.10.2007 17:46 -666
^b2_work	dat	13.868	10.10.2007 15:36 -666

The structure of the data file is as following:

^bx_yyyy.dat

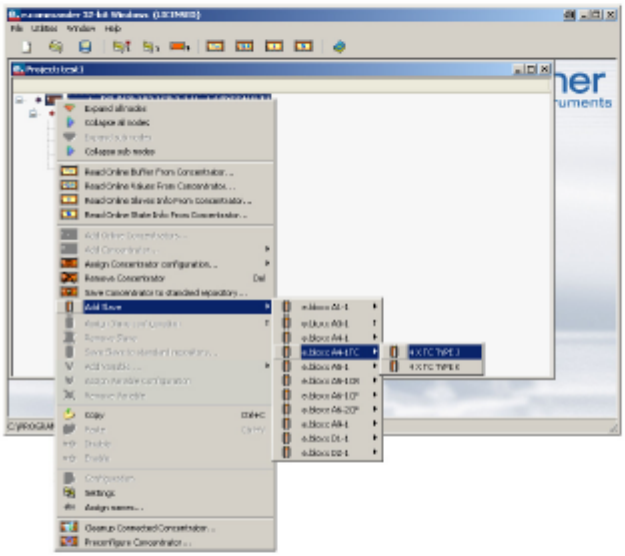
x...index of the data buffer (0..3) according to the data buffer (1..4)
yyyy...index of the file

Within the running system data are being stored to the file ^bx_work.dat
As soon as a data file is being read, all data are being copied to the file ^bx_yyyy.dat which is the file e.g. being transferred to the USB-stick. Next time the file index will be increased by 1. Sending a data file via e-mail or transferring it to an FTP-server will be handled in the same way.

5.7 Adding e.bloxx modules to an e.reader

To add a new e.bloxx sensor module (slave) to an e.reader (the Baud rate of the e.bloxx module has to be changed to 115.200 before as this is the maximum possible Baud rate for slave module, the default Baud rate is 1,5MBit) click with the right mouse button on the icon of the e.reader. From the context menu that appears select "Add Slave" and select the required module with one of the predefined configurations.

Limitation:
Up to 126 single modules (or up to 126 addresses – e.g. an e.bloxx A1-4 has 4 addresses) can be connected to one e.reader. Another limitation is 256 channels in REAL-format or 512 channels in INTEGER-format – or a mixture of these.

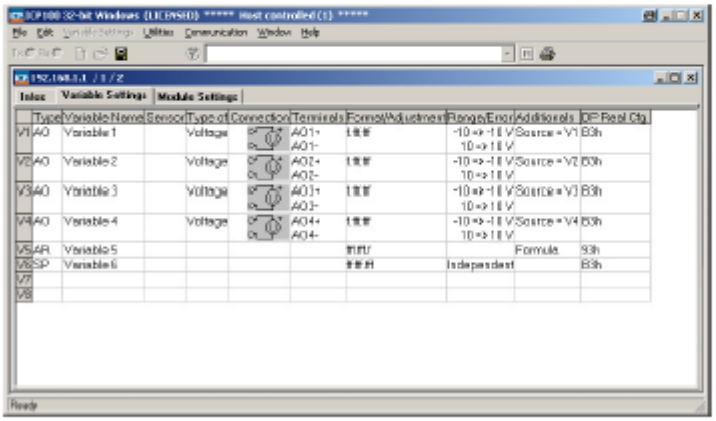


In the following window you have to define the UART interface of the test controller, where the e.bloxx module has to be connected to and the address of the e.bloxx. Since an e.reader only has one slave interface (UART) UART 1 will be selected and cannot be changed.

Now the module is added to the e.reader in the project with standard measurement settings for its in-/outputs.

5.7.1 *Configuration of e.bloxx variables*

Now configure the measurement functions of every variable of the just added e.bloxx. Therefore double-click on the e.bloxx module or a variable or click with the right mouse button and select "Configuration". The Configuration Software ICP 100, which is integrated in test.commander, will be opened. You will see the predefined variable definition.



Every variable is displayed in a row. You can set a name for the variable (name will be shown in the project window) and you can select the type of measurement, the measurement range, format settings and more different settings depending on the type of variable. Click on the corresponding field (column and row) of the setting you want to change. For a more detailed description see the online help of the Configuration Software ICP 100.

After configuration close the ICP 100 Configuration Software by pressing the icon "Save to File and Exit". You will return to the test.commander project window and the variable settings will be updated.

5.7.2 *Limitation of additional e.bloxx modules*

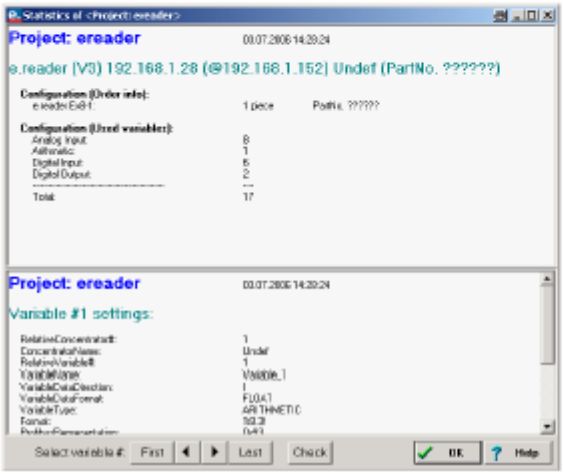
With an e.reader up to 126 additional e.bloxx modules (single modules, resp. addresses) can be added. But nevertheless there is a main limitation which might reduce the number of additional e.bloxx modules.

There is no slot limitation, but the number of data to be read by the e.reader is limited to 1024 Byte.
These are e.g. 256 channels in real format or 512 channels in integer format – or a mixture of both.

5.8 Statistics

Another very useful tool is the statistics functionality. This feature can be accessed when clicking the button in the menu bar.

This functionality provides the following information:



The first part of the window shows an overview on the whole project with:

- Project name, date and time
- IP-address, name/location, part number of the e.reader
- Type and number of e.bloxx modules connected to the e.reader with its part number(s)
- The number of each type of variable like alarm, analog input, analog output,... being defined for this project.

If there are several e.reader in one system, you will find this information for each one.

The second part describes all variables being defined in the project with its most important information in the order as they are configured. With the buttons "First", "Last" and the arrows you can search for the variable you are interested in.

5.8.1 *Design Rule Check*

The performance of the system depends on the numbers of variables, number of variables, transfer rate, data format, active interfaces, etc.

So it is very helpful to check a configuration after the first setup or after having reconfigured the system by clicking the "Check" button.

The result of the test will be explained by the background colors:

- WHITE: Configuration is fine and will run after downloading.
- RED: Configuration will not run, a download is not possible.
- YELLOW: It is not definitely possible to say ok or not ok. In this boarder range it is necessary to test the setup. It is possible to download the configuration.

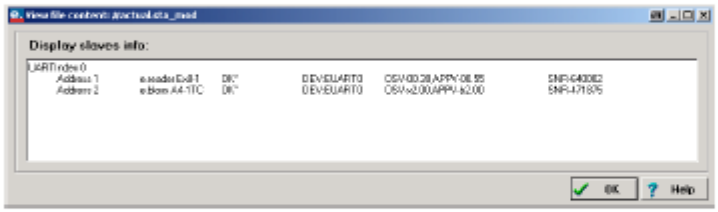
In praxis mostly the error state will be generated by:
- connecting too many modules to the UART.
- defining too many variables.
- selected sample rate could not be reached.

5.9 Display of Slave Information

Slaves are the e.bloxx measurement modules connected to the e.reader. Information about those slaves can be displayed on screen.

To read out and display the slave information click on the icon "Read Online Slaves Info from Concentrator". The test.commander will scan the network and display all connected concentrators in a window (see chapter 5.2).

Select the desired e.reader and press "OK". The e.reader will be read out and the information about the slave modules connected to the e.reader will be displayed in a new window.
Note: Because of the hardware structure of the e.reader the I/O part will be shown as slave module e.reader Ex8-1 as well – its address always is 1.

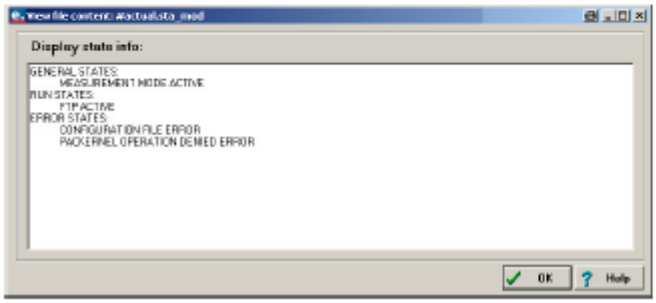


5.10 Display of Status Information from Concentrator

The actual state of an e.reader can be read. These information include run states, error information and general states like if the measurement mode is active, etc.

To read the status information from an e.reader click on the icon "Read Online State Info from Concentrator". The test.commander will scan the network and display all connected test controllers in a window (see chapter 5.2).

Select the desired e.reader and press "OK". The e.reader will be read out and the status information will be displayed in a new window.



6. FUNCTIONALITY WITH TEST.CON

The e.reader permits complex control functionality such as PID controller, state machine, arithmetic, numeric and logical operations, transmission terms, function generators, time functions as well as visualization and additional functionality.

The software test.con is an easy-to-handle graphical programming system to define the functionality of an e.reader. As only graphical blocks are used for the definition, time to get familiarized with the system is minimized. The presentation of graphical blocks follows existing specification and technical languages. Structure blocks permit a hierarchic structure of projects. Linked projects and export mechanisms facilitate the reuse of structure blocks that have been created earlier.

The software test.con combines programming, simulation, testing and running in one tool. Special blocks and additional tools permit online observation of signals and signal tracing as well as run-time measurements.

Note: This chapter will give a short overview of the test.con software. A more detailed description can be found in the test.con software manual and the online help of test.con.

6.1 Overview of the test.con User Interface

After in installation of test.con from the CD start the software and the main window of test.con will open.

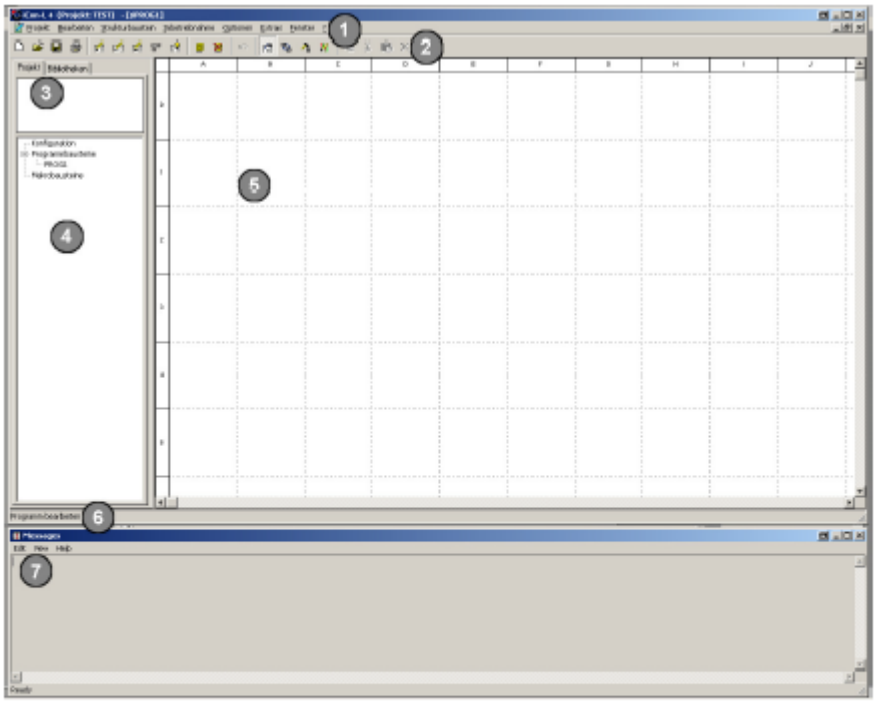


Figure 6.1 - Main Window of test.con

Commands for working with the system can be selected from the Menu ①, the Toolbar ②, the Quickstart Window, and Context Menus as well as by using Keyboard Shortcuts. The structure of the project is presented in the Project Tree ④ which is displayed when you click on the tab "Project" on top of the tree. A number of operations can also be carried out from the project tree. By selecting a structure block in the project tree a graphical preview of it will be shown in the preview area ③. From there it is possible to drag the structure block into the working area ⑤, which contains the worksheet with the presentation of the blocks and their connections. Instructions, status information and messages are displayed in the Status Line ⑥ and the Messages Window ⑦.

The whole functionality of the function blocks of the system is defined in the e.reader. The software test.con is just a tool for combining and "wiring" the single blocks.

6.2 Projects

A project can be subdivided into any number of hierarchic structures. However, it always comprises not less than two levels, i.e. its configuration and program blocks. A project is processed by arranging the desired structure blocks (program and macro blocks) and connecting them graphically.

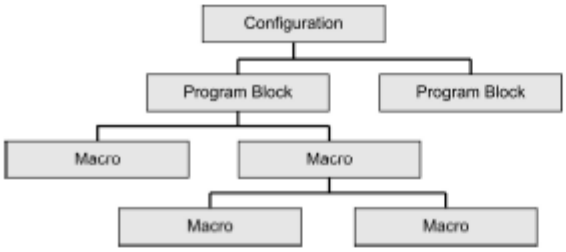


Figure 6.2 - Structure of test.con

The top level of a project is the configuration (&Main). It allows the graphical configuration of the target system tasks.

Program blocks are added to the configuration. They form the second level of the project where the functions are called and their sequence is established. Program blocks are flagged with a $ symbol preceding its designation. By determining the priority and cycle time of program blocks the task management parameters are defined. Connections between the program blocks facilitate the asynchronous data exchange via the e.reader. The number of program blocks is limited by the number of the parallel tasks in the target system (usually 15).

Programs are structured by using macro blocks. They make it easier to call functions which are used repeatedly. Macro blocks can be inserted into program blocks and other macro blocks. However, recursive calling is not permitted. Macro blocks contain sub functions of the programs and relating local parameters. These are no subprograms. During code generation they generate the complete target code for each call.

All project levels may contain visualization blocks (without target function). For exchanging data between project levels, the Input and Output blocks from the Standard Library are used. The Enable block from the same library is used for conditional processing operations of program and macro blocks.

6.3 Classes and Instances

When a project is processed, there is a difference between classes and instances. Classes are generated during editing operations by combining certain structure blocks and they contain preset values. Instances are formed when changing to the "Run" mode and can be parameterized there. So it is possible to define a special functionality (class) in the "Edit" mode which can be used in different programs/macros with different parameters.

During editing operations only preset values are defined in the parameter dialogs of the blocks. During the first change to the *Run* mode they are copied into instance parameters. Thereafter, the instance parameters can be modified in the *Run* mode. However, these modifications have no effect on the preset values in the structure block classes. When returning to *Edit* and modifying preset values, the instance parameters remain unaffected. The instances of newly inserted structure blocks contain the preset values as instance parameters.

Note: When changing a parameter in the *Run* mode (instance parameter) it is necessary to download them into the e.reader to ensure that the new parameter is still valid in the e.reader after a power off/on, otherwise the old parameter are loaded after a new start.

To facilitate easier differentiation between structure block instances, they can be assigned to different names. The context menus for structure blocks in the *Run* mode contain the *Instance Name* command. Upon opening the dialog a designation may be entered which afterwards in the symbol of the structure block and in the project tree.

Instance names are only indicated in the *Run* mode. An exception is the configuration where the instance name of the programs remains visible even during editing operations.

6.4 The Four States of the test.con Software

The software test.con facilitates creating, changing, testing and running projects by using one tool only. The program distinguishes between the following 4 system states:

- Edit
- Design
- Run
- Online Observation

Each state can be obtained by using the menu, the toolbar or partly the context menu.

6.4.1 Edit State

"Edit" is the basic state of the system. It is the state where a project is mainly processed. The structure of a project is determined by loading and removing libraries, defining and deleting structure blocks, and inserting, removing and connecting structure blocks. In the edit mode the classes of structure blocks are processed. The values entered in the parameter dialogs of the blocks only serve as presetting for the instances visible in the run mode.

Upon entering one of the first three commands of the edit menu or closing all work sheet windows, the system changes to the edit state. This state can be easily identified through the appearance of the project tree. There, the defined structure blocks and loaded libraries are listed in two views. In addition, the grid for block positions and connections are indicated by means of dotted lines in the work sheet windows.

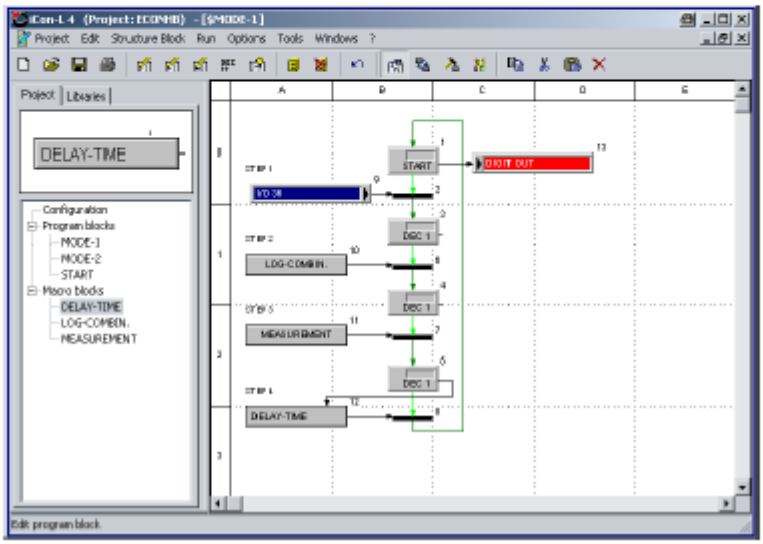


Figure 6.3 - Worksheet Edit State

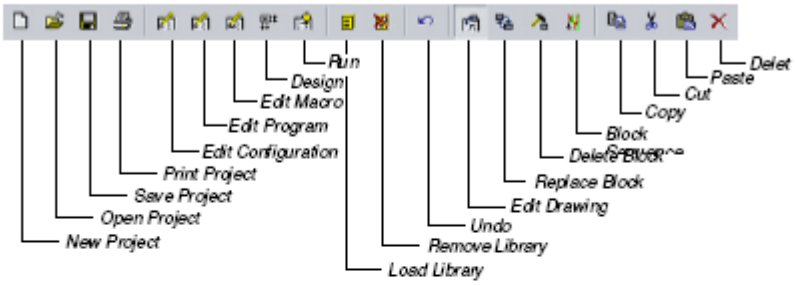


Figure 6.4 - Toolbar Edit State

6.4.2 Design State

Apart from the operations described above, i. e. inserting and deleting objects, also the structure block design ("Design Structure Block") can be used. From a specific work sheet window and by means of an additional toolbar, label positions, the appearance of symbols and the presentation in the work sheet window can be changed.

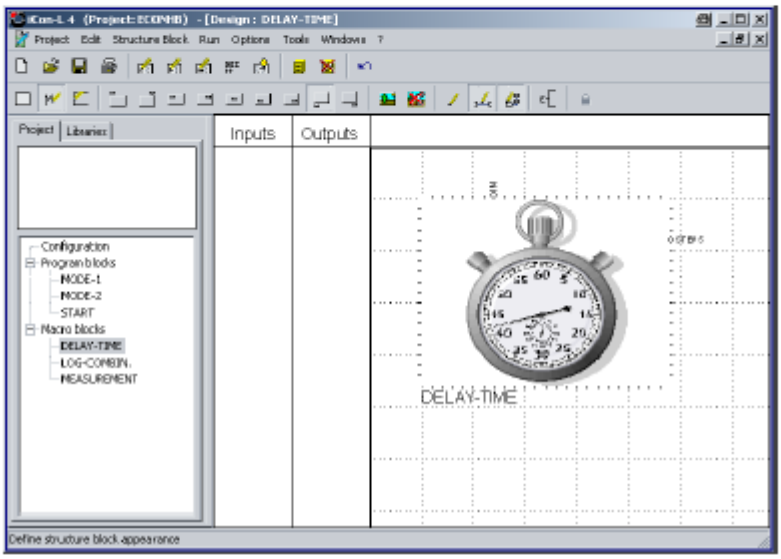


Figure 6.5 - Worksheet Design State

Additionally to the already mentioned toolbar there are some more functions available to design the appearance of the project, the programs and the macros.

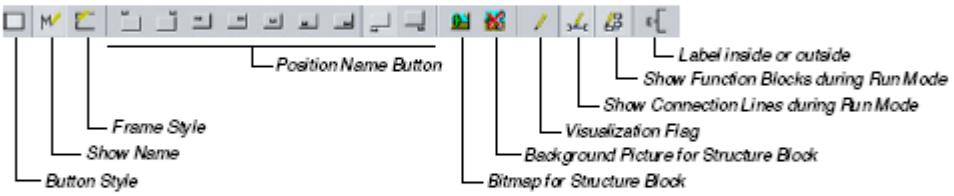


Figure 6.6 - Toolbar Design State

6.4.3 Run State

When using the Enter command from the "Run" submenu the program turns from the edit to the run state. During this changeover the program performs an instantiation. Instances are formed from the classes established during editing operations (refer to chapter 6.3 - Classes and Instances). When this option is chosen, a new work sheet window is opened, indicating the configuration. In contrast to the work sheet windows in the edit mode, the work sheet windows in the Run state does not contain a grid. The project tree offers one view only, presenting the hierarchic structure of the project.

The run mode serves to assign instance parameters used by the blocks later on in the e.reader. Commands for importing and exporting parameter files (refer to the submenus "Structure Block" and "Run") support this work. In addition, cycle times of tasks can be defined. In the run state structural changes of the project are no longer possible. The other commands in the run submenu serve to start simulation, establish target system connections as well as to create, load and start the program and change to online observation.

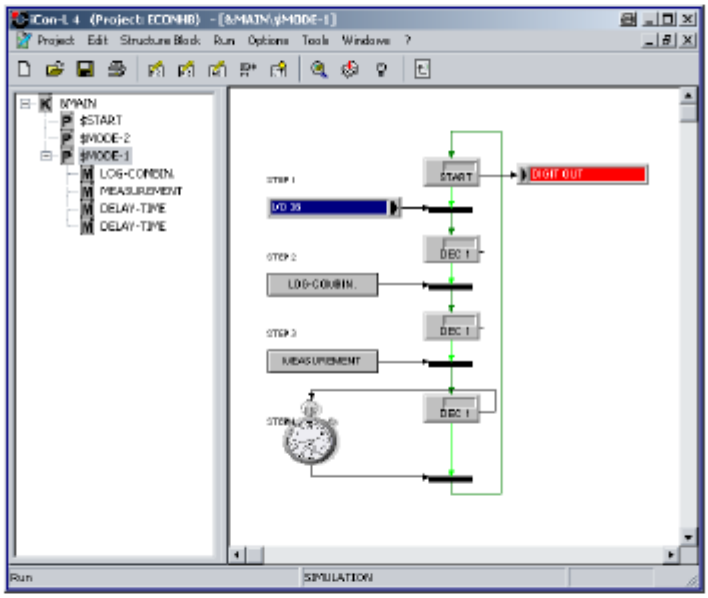


Figure 6.7 - Worksheet Run State

The toolbar of the run state is similar to that one of the edit state. The difference is shown in the next picture.

Figure 6.8 - Toolbar Run State

6.4.4 *Online Observation State*

When a program has been successfully loaded into simulation or e.reader, online observation starts automatically. For switching from the run mode without performing a download before directly, the "online" command in the "Run" submenu is being used.

For online observation the visualization blocks indicate the current values of connected signals. Clicking the connection lines by means of the left mouse button opens a window displaying the current signal value. Via parameter blocks and signal editors values are being changed online. However, they only influence the volatile memory (RAM). For transferring the values into the nonvolatile memory of the e.reader, the "Parameter" command in the "Logon Target System" dialog is used.

If the project uses archive blocks, they record signal patterns in the e.reader. Signal patterns can be read out and displayed by means of an additional tool iTrend.

The refresh rate of value representation can be defined in the "Work Sheet Settings" dialog. The online observation can be stopped by using the "Offline" command/button or by exiting test.con.

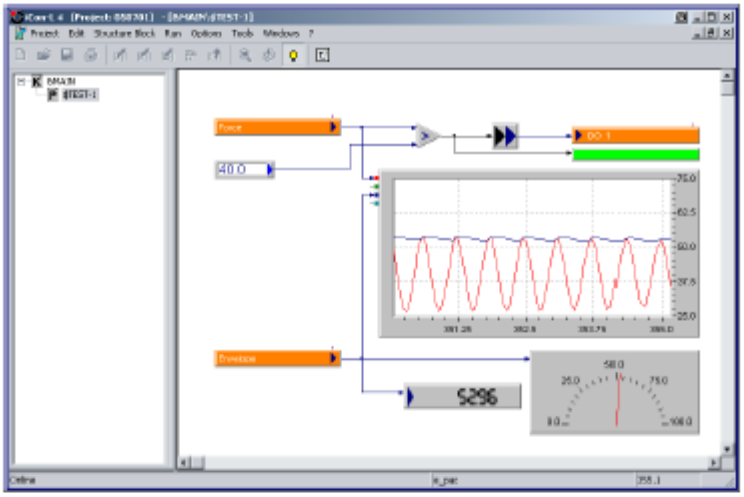


Figure 6.9 - Worksheet Online Observation State

6.5 Starting a Project

After the program test.con had been started it is possible

- to start a new project,
- to open an existing project,
- to upload the project from a connected e.reader or
- to load a project from the restore directory.

Start a new project:
Click on the button *Create a New Project* or select the menu *File - New Project* and the following window will appear:

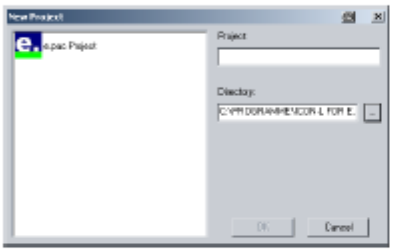


Figure 6.10 - Window New Project

Enter a project name and confirm/modify the directory where the project shall be stored.

Open an existing project:
Select the project and open the xxx.MDL file

Figure 6.11 - Window Open Project

Upload a project from e.reader:
The window for the selection of the so called *target system* will open. There all selected e.reader modules and the PC-simulation are offered. By clicking the relevant e.pac the color will change from red to green.

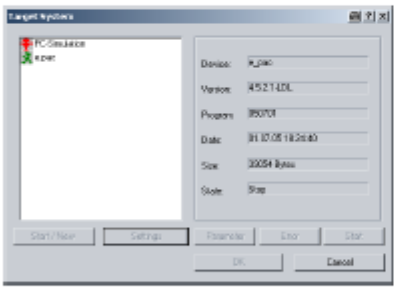


Figure 6.12 - Window Upload from e.reader

In case the icon color is yellow the settings (IP address) have to be checked.

Load a project from restore directory (zipped file):
Selection of the file to be opened.

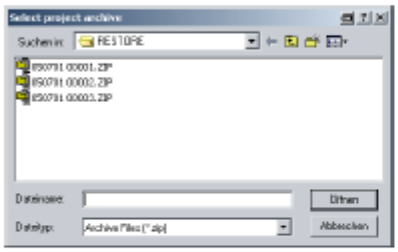


Figure 6.13 - Window Load from Restore file

6.6 Connecting e.reader and test.con

In a new or an already existing project open the test.con has to be selected first by using the menu *Tool - Select Device*: Now it is possible to select *Online* or *Offline*. In case *Offline* is selected a browser will open to select a #summary.sta file.

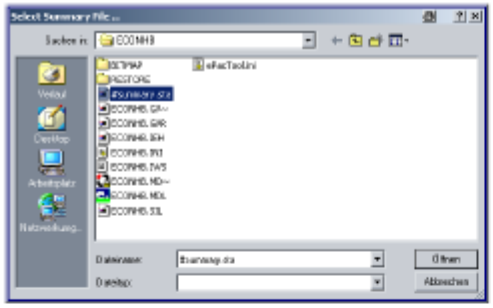


Figure 6.14 - Selection Online/Offline and Browser to choose a #summary.sta file

Selecting *Online* a network scan will be done. After selecting the e.reader the successful selection has to be confirmed.

Figure 6.15 - Successful selection of an e.reader

Now the software test.con will work with all the variables defined in the measuring and I/O system.

6.7 Practical Use of the Structuring

In the practical use the combined measurement, automation and control tasks consist of many sub tasks, which themselves consist of different sequences, flow parameter and evaluations resp. decisions. This chapter gives an introduction on how to layout a project. More information can be found in the online help of test.con and the test.con instruction manual.

In a first step we recommend the definition of the process inputs and the process outputs. This could be done in a macro "Inputs" and in a macro "Outputs". Due to the use of global parameter these signals will then be available in any further program and macro block.

In the next step the total function of the project should be partitioned in logical function such as Test Mode 1 and Test Mode 2. The block "START" will enable/disable the different Test Modes.

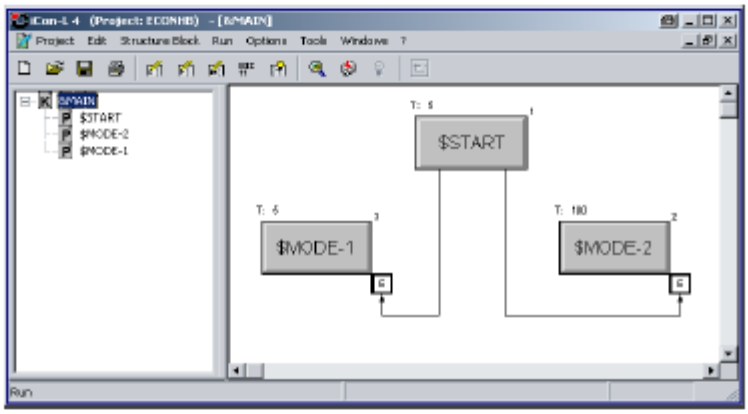


Figure 6.16 - Example: Project with Test Mode 1 and Test Mode 2

A Test Mode consists of different steps; each step can be controlled by a process I/O (host or process controlled) or by a decision made in the e.pac. The program MODE-1 could contain e.g.:

Step 1: Proceed at an external I/O signal
Step 2: Proceed by a combination of events which have happened
Step 3: Proceed when a measurement signal exceeds a set point limit
Ste4 4: Proceed after a special time (delay)

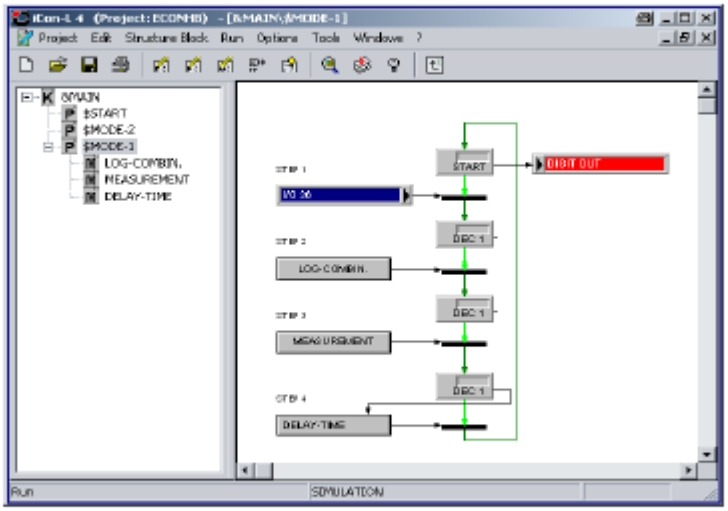


Figure 6.17 - Example: Function of Test Mode 1

In the example the macros LOG-COMBIN, MEASUREMENT and DELAY TIME have the following function:

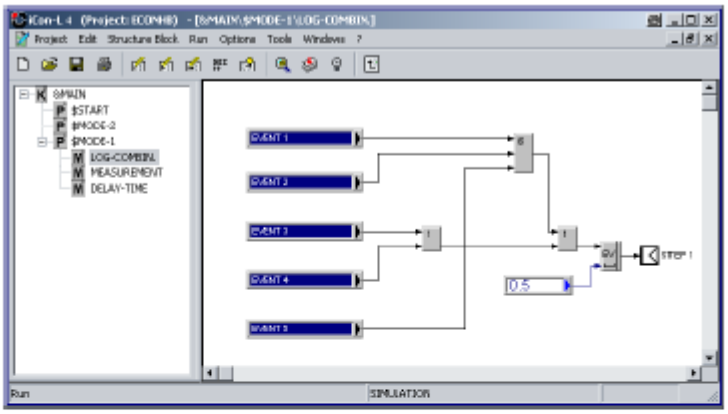


Figure 6.18 - Example: Function of macro LOG-COMBIN

Providing the events according to the logical combinations as a "true" signal at the output *STEP 1*, the condition is fulfilled and the next condition is required.

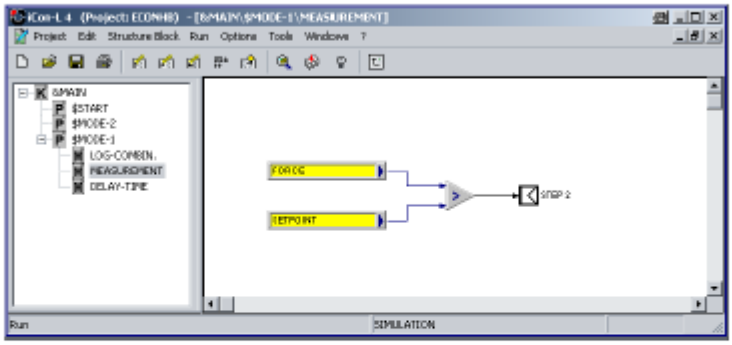


Figure 6.19 - Example: Function of macro MAESUREMENT

If *FORCE* exceeds the *SETPOINT* the output of *STEP 2* will be "true" and the next macro is in charge.

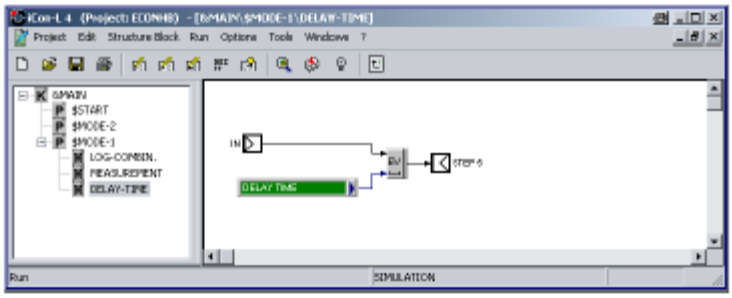


Figure 6.20 - Example: Function of macro DELAY-TIME

After a selectable *DELAY-TIME* the input will be switched to the output *STEP 3* and all three conditions are fulfilled.

This example shows how a structure can be used to setup a clear functionality by using different program and macro

7. HARDWARE INSTALLATION

7.1 Environmental Conditions

The e.reader is protected against water and dirt according IP 20. If required by the conditions of the operating site the modules have to be installed accordingly, e.g. in a water-resistant or water-proof case, compliant with the regulations of electrical engineering.
For the allowed ambient temperatures for the e.reader see the Technical Specifications at the end of this manual.

7.2 Mounting

The e.reader has a snap-on mounting for installation on standard profile rails 35 mm (1.4 inch) according to DIN EN 50022. The mounting on the DIN rail is performed by four straps on the reverse side of the module. First you push the two lower straps behind, the lower notch of the DIN rail and then you press the module on the DIN rail until the two upper straps snap in to the upper notch.

In order to take the module off the DIN rail either slide the module side-wards off the rail or, if it is not possible, lift the module slightly so that the straps on the top are released from the notch and the module can be taken off easily by tipping it forwards and removing it from the DIN rail.

7.3 Module Parts

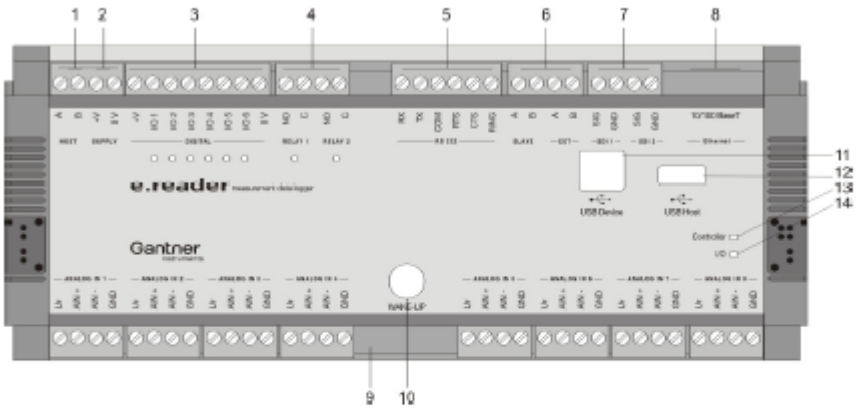


Figure 7.1. – Module Parts of the e.reader

1 to 7 and 9 ... Pluggable screw-type terminal strips:
- 1 .... RS 485 serial bus for connection to a PC/laptop
- 2 .... Power Supply +10 up to 30 VDC
- 3 .... 6 Digital I/Os
- 4 .... 2 Relay outputs
- 5 .... RS 232 serial bus for connection to a PC/laptop (us the RS 232 cable included in the scope of supply)

6 ... Slave bus for connecting other e.bloxx measurement modules and external devices
7 ... Two SDI interfaces
8 ... RJ 45 plug for Ethernet connection
9 ... Eight analog inputs
10 .. Push button to weak up the processor unit
11 .. USB Device interface
12 .. USB Host interface
13 .. Status LED "Controller"
14 .. Status LED "I/O"

7.3.1 Front LEDs

The LED at the front of the e.bloxx modules provides the following information:

LED "Controller": This LED indicates the status of the processor board.
LED "I/O": This LED describes the status of the I/O board of the e.reader.

7.4 Connection Technique

The wires for the power supply, serial interfaces RS 232, RS 485, SDI and for the sensor signals are connected to the e.reader via screw-type terminals. The captive terminal screws are part of the terminal strips. All terminal strips are of a plug-in type and can be detached from the e.reader.
Not more than 2 leads should be connected with one clamp. In this case both leads should have the same conductor cross section. For the precise clamping of stranded wire we recommend the use of wire-end ferrules.

Notice: Connecting wires respectively the plugging-in and -out of the terminal strip is only allowed with an e.reader in power-off status.

In order to prevent interference with sensors, signals and modules, shielded cables have to be used for the power supply, bus connection and signal lines.

For the connection of the Ethernet a standard RJ 45 connector is provided by the e.reader.

7.5 Power Supply

Non-regulated DC voltage between +10 and +30 VDC is sufficient for the power supply of the modules. The input is protected against excess voltage and polarity connecting error. The power consumption remains approximately constant over the total voltage range, due to the integrated switching regulator.

Due to their low current consumption the modules can also be remotely supplied via longer lines. Several modules can be supplied in parallel within the permissible voltage range and drop in the lines. If required, the supply lines together with the bus line may be incorporated in one cable.

In order not to overload the module power supply needlessly and to avoid unnecessary line troubles, a separate power supply is recommended for sensors with a large current drain.

The distribution voltage for the e.reader modules has to be protected by a fuse with maximum 1 A (inert). The modules have an internal fuse (reversible) for protection against excess voltage, excess current and wrong polarity.

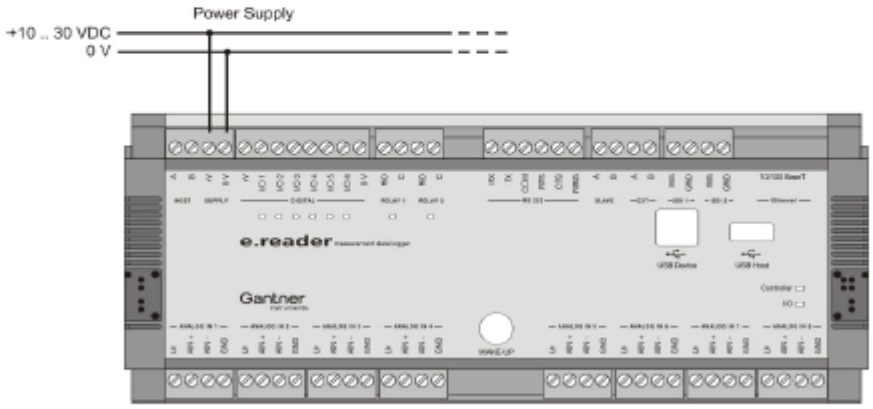


Figure 7.2 - Power Supply of the e.reader

7.6 Connection of e.bloxx Modules

e.bloxx modules are connected to the RS 485 slave interface (screw terminals "A$_{Slave}$" and "B$_{Slave}$") at the e.reader.

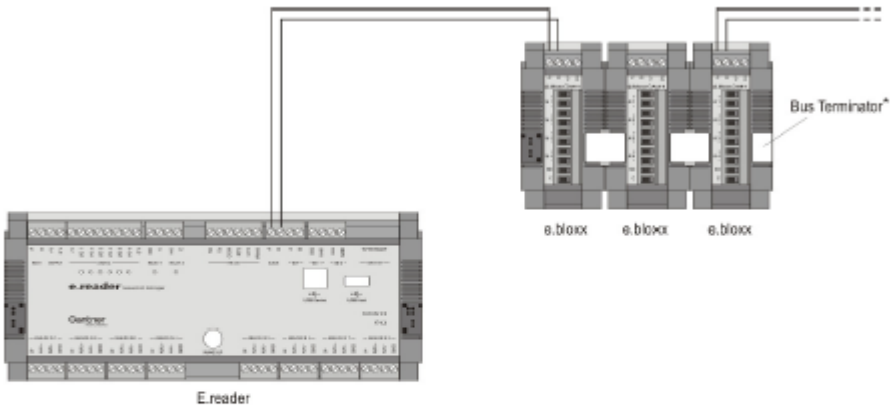


Figure 7.3 – e.bloxx Modules connected to e.reader

* ... If the e.bloxx (here A4-1TC as an example) are used together with an e.reader, which is used to collect the data of all connected e.bloxx modules and process them for fast transmission via the further network, a bus termination must be connected to the last e.bloxx in each bus line.

7.7 Connection to Ethernet

The e.reader is supported with a connector 10/100 BaseT to connect the e.reader into an Ethernet. The IP address and DHCP setting must be set-up correctly (see chapter 3 "Start-Up"). Then it is possible to communicate with the e.reader via the ethernet.

It is also possible to connect the e.reader directly to a PC via the 10/100 BaseT connector. In this case a cross-wire Ethernet cable must be used.

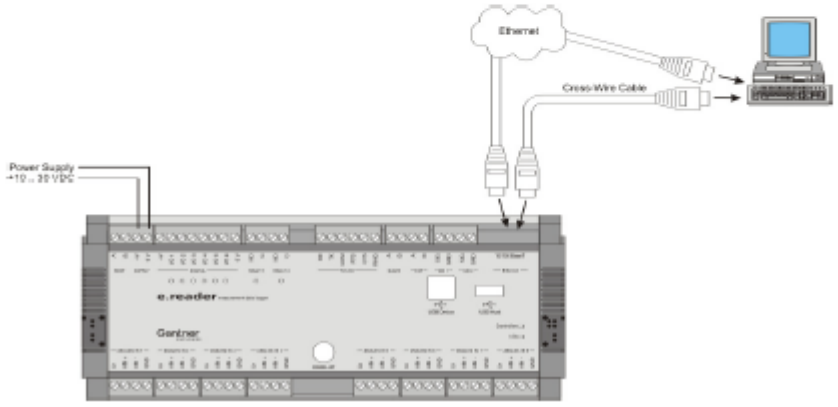


Figure 7.4. - Connection to Ethernet

7.8 Connection to the RS 485 Communication Bus

The e.reader has an RS 485 serial interface. The bus lines are connected to the screw terminals "A_{HOST}" and "B_{HOST}" of the e.reader.

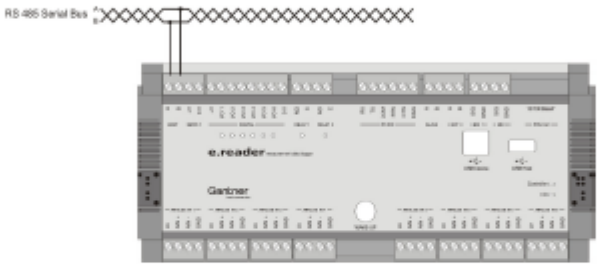


Figure 7.5. - Connection to RS 485 Serial Interface

A detailed description of the bus and the communication with the e.reader and the connected e.bloxx modules can be found in the manual "e.bloxx Communication" (Part No. 159583).

7.9 Shielding RS 485

In case of increased interference, such as in industrial areas, we recommend a shielding of the power supply, bus, and signal cables. In general, the shield should be connected to the protective earthing (not Data ground!) at each bus connection. If necessary, the shield should also be applied along the course of the cable several times. For shorter distances, e.g. with stub cables, the interference response is often improved if the shielding is only applied to the stub cable exit.

Bus users such as controllers (PLCs), computers (PCs), repeaters and interface converters (ISK), etc., generally feature the possibility of applying the shield directly to the appliance or to separate shield rails. Shield rails offer the advantage of preventing possible interfering signals from reaching the appliance. The shields which are connected to protective earthing conduct interference signals off before reaching the module.
The e.reader does not have a direct shield connection at the module. Here the shield of the bus cable can be connected to earth e.g. by so-called shield clamps.

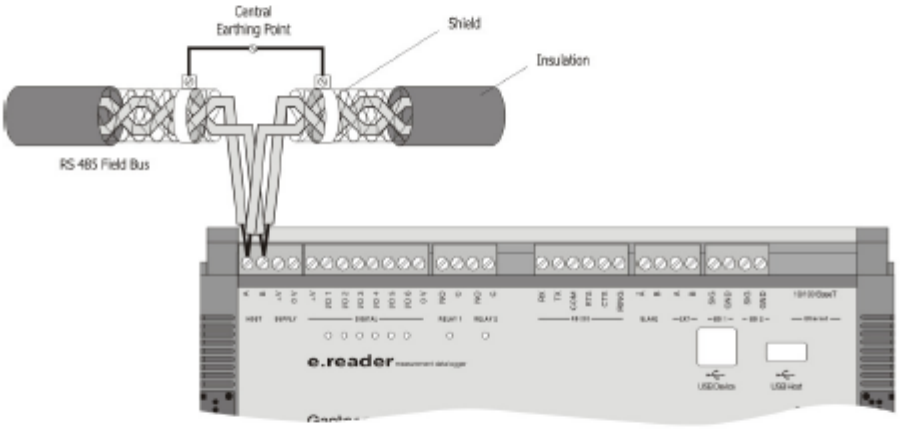


Figure 7.5. - Earthing of the Bus Line Shield at an e.reader

Notice: The shielding screen must not be connected to the ground (0V) of the power supply and it should always be connected to earth with a large surface and low-inductance.

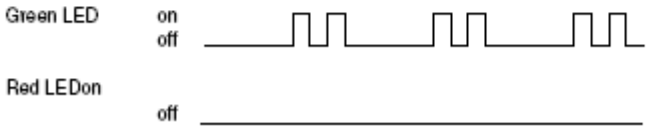
7.10 LED Indication

7.10.1 Indication of RUN and ERROR States

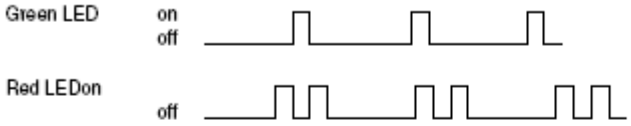
The lower one of the LED indicates the RUN and ERROR states. This LED will shine red and/or green

Each different state increases the number of blinks. One blink means that the LED is switched on for 200 ms (20 ms with the USB connection) and off for 200 ms (20 ms with the USB connection). At the start of an indication sequence there will be a period of 1000 ms during which the LED is switched off.

Example:
If there are e.g. 2 active RUN conditions and no ERROR, the LEDs blinks like this:
Green LED: 1000 ms OFF -> 200ms (1st) ON -> 200ms (1st) OFF -> 200ms (2nd) ON -> 200ms (2nd) OFF restart
Red LED: OFF



If there are e.g. 1 active RUN condition and 2 ERROR, the LEDs blinks like this:
Green LED: 1000 ms OFF -> 200ms ON -> 200ms OFF -> restart
Red LED: 1000 ms OFF -> 200ms (1st) ON -> 200ms (1st) OFF -> 200ms (2nd) ON -> 200ms (2nd) OFF restart



RUN states:

- DATAPORT ACTIVE	... TCP/IP data port activated
- PACKERNELPORT ACTIVE	... e.pac kernel port activated
- HIGHSPEEDPORT UDP ACTIVE	... UDP highspeed port activated
- HIGHSPEEDPORT TCP/IP ACTIVE	... TCP/IP highspeed port activated
- DISTRIBUTORPORT ACTIVE	... UDP distributor port activated
- FTP ACTIVE	... FTP activated
- CONFIGBUS RS232 ACTIVE	... RS 232 configbus activated
- CONFIGBUS RS485 ACTIVE	... RS 485 configbus activated

ERROR states:

- SLAVE ERROR	... Slave error (not found, data type mismatch,...)
- SLAVE LINE x INSTABLE ERROR	... Bad line
- CONFIGURATION FILE ERROR	... Error in configuration file(s)
- DATA FLASH FILE SYSTEM ERROR	... Data FLASH error (full,...)
- REDUCED PERFORMANCE ERROR (1000.0 Hz -> 500.0 Hz)	... due to health is too low
- PACKERNEL OPERATION DENIED ERROR	... due to e.pac kernel operation is denied (e.g. configuration has changed)
- DISTRIBUTORPORT SYNC ERROR	... due to distributor port thread has too less time to be synchronized to the slave communication
- SOCKET OVERLOADED ERROR	... due to socket has no time to work proper
- EXTENSION BOARD ERROR	... due to unknown extension board found

8. SPECIFICATIONS

All following data are valid after a warm-up time of approx. 45 minutes.

8.1 Analog Inputs (8 per module)

Accuracy	0.01 % up to 0,5% range dependent
Repeatability	0.003 % typical (within 24 hours)
Type of Measurement	Range
Voltage	±10 V
	±5 V
	±1 V
	±100 mV
	±10 mV
Current	25 mA
	1 mA
imput impedance	100Ω
Resistance 2-/4-wire	20 kΩ
	4 kΩ
	400 Ω
Measurement current	0,5 mA DC
Linearity deviation:	0.01% of the final value
Temperature influence	
on zero	10 µV / 10 K
on sensitivity	0.02 % / 10 K

8.2 Analog/Digital Conversion

Resolution	16 bit
In numbers	0,003 up to 0,01% range dependent
Sample Rate	1 s up to 24 h
Conversion method	Sigma Delta

8.3 Digital In- and Output (6 per module)

Input	
Functionality	Status, counter, frequency
Input voltage	max. +30 VDC
Input current	max. 1,5 mA
Input frequency	1 kHz
Output	
Functionality	Process- or host controlled
Type of output	Open-Collector
Output voltage	max. 30 VDC
Output current	max. 100 mA
Output frequency	max. 100 Hz

8.4 Relais Outputs (2 per module)

Functionality	Closer
Current	max. 1 A
Voltage	max. 60 VDC

8.5 Data Memory

Flash (non volatile)	128 MByte
RAM (volatile)	8 MByte
Logging Interval	1 s up to 24 h individual per channel

8.6 Interfaces

RS 485	1 Slave
	1 Master and Slave
	1 Master for connection of additional modules
RS 232	1
USB Client	1
Ethernet TCP/IP	1
SDI-12	2
Data Formats:	8E1, 8N1 selectable
Protocols:	ASCII, MODBUS-RTU (parts)
	Gantner LocalBus (binary), PPP
As master	Ott, SD-12, DDP (in extracts)
Baud Rates:	2.4it/s up to 115.2 kbit/s
	LocalBus up to 1.5Mbit/s

8.7 Power Supply

Power supply	10 VDC to 30 VDC over voltage and overload protection
Power consumption at	
Samplerate of 10min	approx. 100 mW
Samplerate of 1min	approx. 200 mW
Samplerate of < 20s	approx. 4 W

8.8 Mechanical

Case:	Aluminium and ABS
Dimensions (W x H x D):	225 x 90 x 83 mm (8.8 x 3.5 x 3.3 inch)
Weight:	800 g
Protective System:	IP 20
Mounting:	DIN EN-Rail

8.9 Connection

Plug-In screw terminals	Wire cross-section up to 1.5 mm²
Ethernet	RJ 45 plug

8.10 Environmental Conditions

Operating Temperature	-20 °C to + 60 °C (-4 °F to + 140 °F)
Storage Temperature	-30 °C to +85 °C (-22 °F to + 185 °F)
Relative humidity	0% up to 95% at +50 °C (+122 °F), non-condensing

8.11 Electromagnetic Compatibility

Electro static discharge (ESD)	level 2 acc. IEC 801-2: 4 kV
Radiated electromagnetic Fields	level 3 acc. IEC 801-3: 10 V/m
Electrical fast transients	level 3 acc. IEC 801-3: 2 kV / 1 kV
Radiated RFI/EMI	level B acc. VDE 0871-1/CISPR 11

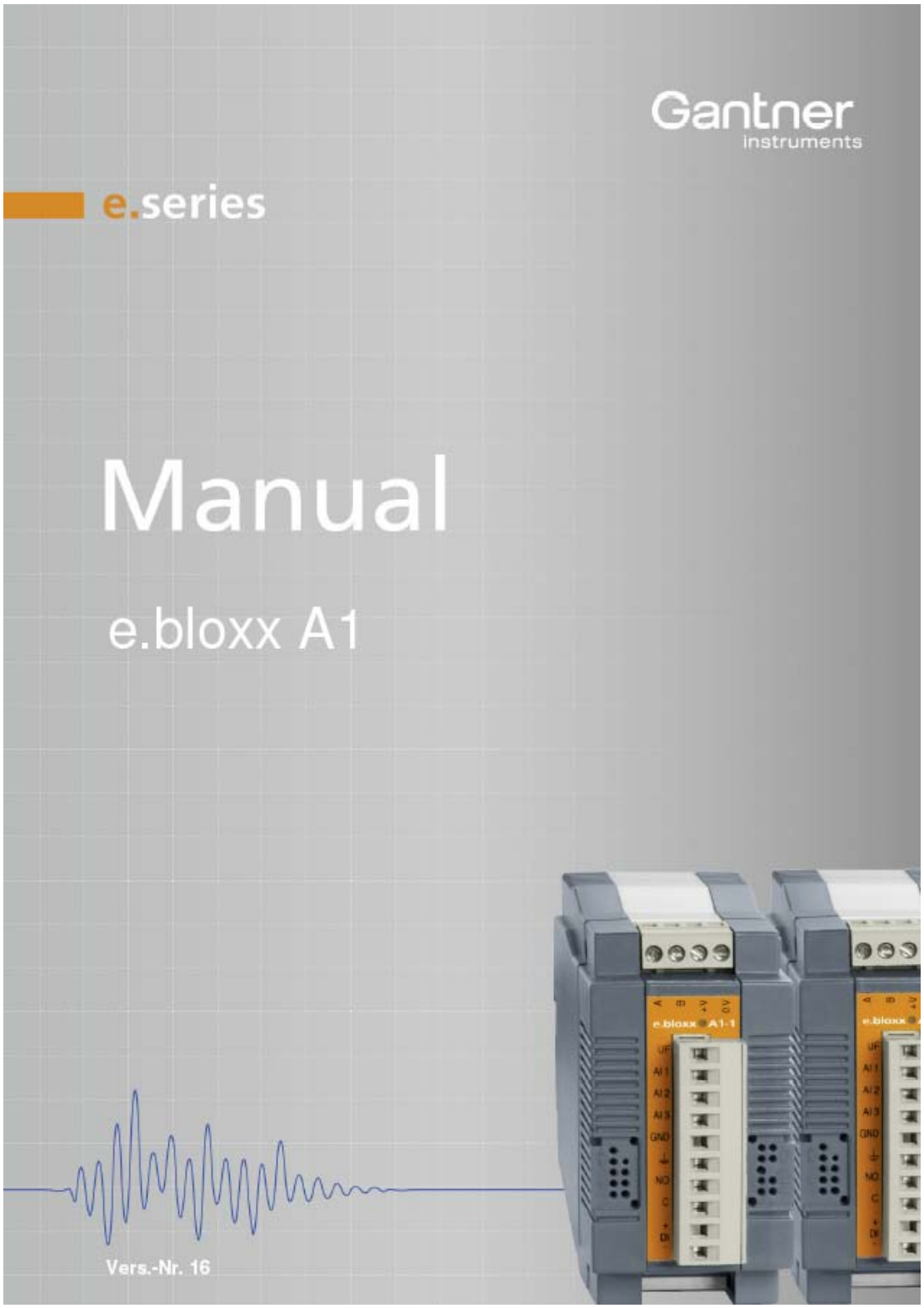
Part No. of e.reader: 511373

www.gantner-instruments.com

Gantner Instruments Test & Measurement GmbH
Montafonerstraße 8 • A-6780 Schruns/Austria
Tel.: +43 (0)5556-73784-410 • Fax: +43 (0)5556-73784-419
E-Mail: office@gantner-instruments.com

Gantner instruments

2. Módulo de entradas/salidas E.Bloxx A1-8-



Important: Before commencing installation, commissioning, putting into service and before any maintenance work is carried out, it is essential that the relevant warning and safety instructions in this manual are read!

General warning and safety instructions:

Dear customer,

We congratulate you on having selected a product of Gantner Instruments Test & Measurement GMBH. So that our product functions in your installation with safety and to your complete satisfaction, we take this opportunity to familiarize you with the following ground rules:

1. Installation, commissioning, operation and maintenance of the product purchased must be carried out in accordance with instructions, i.e. in accordance with the technical conditions of operation, as described in the corresponding product documentation.

2. Before installation, commissioning, operation or maintenance it is therefore essential that you read the corresponding chapter of this manual and observe its instructions.

3. If there are still some points on which you are not entirely clear, please do not take a chance, but ask the customer adviser responsible for you, or ring the Gantner Instruments Test & Measurement GMBH hot line.

4. Where not otherwise specifically laid down, appropriate installation, commissioning, operation and maintenance of the appliance is the customer's responsibility.

5. Directly on receipt of the goods, inspect both the packaging and the appliance itself for any signs of damage. Also check that the delivery is complete (-> accessories, documentation, auxiliary devices, etc.).

6. If the packaging has been damaged in transport or should you suspect that it has been damaged or that it may have a fault, the appliance must not be put into service. In this case, contact your customer advisor. He will make every effort to resolve the problem as quickly as possible.

7. Installation, commissioning and servicing of our appliances must only be carried out by suitably trained personnel. In particular, correspondingly qualified specialists may only make electrical connections. Here, the appropriate installation provisions in accordance with the relative national Electrical Engineers construction regulations (e.g. ÖVE, [Austrian] VDE, [German]...) must be observed.

8. Where not otherwise stated, installation and maintenance work on our appliances is exclusively to be carried out when disconnected from the power supply. This applies in particular to appliances that are normally supplied by low-tension current.

9. It is prohibited to make alterations to the appliances or to remove protective shields and covers.

10. Do not attempt yourself to repair an appliance after a defect, failure or damage, or to put it back into operation again. In such cases, it is essential you contact either your customer adviser or the Gantner Instruments Test & Measurement GMBH hot line. We will make every effort to resolve the problem as quickly as possible.

11. Gantner Instruments Test & Measurement GMBH accepts no responsibility for any injuries or damage caused as a result of improper use.

12. Although every care is taken and we are continuously aiming for improvement, we cannot exclude completely the possibility of errors appearing in our documentation. Gantner Instruments Test & Measurement GMBH therefore accepts no responsibility for the completeness or the accuracy of this manual. The right is reserved to make alterations, and we may carry out alterations at any time without giving prior notice.

13. Should you discover any fault with the product or in its accompanying documentation, or have any suggestions for improvement, you may confidently approach either your customer adviser or Gantner Instruments Test & Measurement GMBH directly.

14. However, even if you just want to tell us that everything has functioned perfectly, we still look forward to hearing from you.

We wish you a successful application of our appliances. We will be pleased to welcome you as a customer again soon.

Contact address / manufacturer:

Gantner Instruments Test & Measurement GmbH
Montafonerstrasse 8
A - 6780 Schruns/Austria
Tel.: +43 5556 73784 - 410
Fax: +43 5556 73784 - 419
E-Mail: office@gantner-instruments.com
Web: www.gantner-insturments.com

Gantner Instruments Test & Measurement GmbH
Industriestraße 12
D-64297 Darmstadt
Tel.: +49 6151 95136 - 0
Fax: +49 6151 95136 - 26
E-Mail: testing@gantner-instruments.com
Web: www.gantner-instruments.com

TABLE OF CONTENTS

1. ABOUT THIS MANUAL 6

2. MODULE DESCRIPTION 7
2.1. System Overview 7
2.2. Types of Modules 8
2.3. Module Parts 9
2.4. Functional Overview 10
2.5. Front-LED 10
2.6. DC-Isolation 10
2.7. Functional Diagram 11

3. MOUNTING E.BLOXX AND CONNECTING WIRES 12
3.1. Environmental Conditions 12
3.2. Connection Technique 12
3.3. Power Supply 12
3.4. Bus Connection 13
3.5. Shielding 18
3.6. Rapid Bus Link Plug 18

4. MEASUREMENTS 20
4.1. General 20
4.2. Analog Input - Measurement of Voltage 21
4.3. Analog Input - Measurement of Current 22
4.4. Analog Input - Measurement of Resistance 23
4.5. Analog Input - Measurement with a Resistance Bridge 24
4.6. Analog Input - Potentiometric Measurement 25
4.7. Analog Input - Measurement of Temperature with Thermocouples 25
4.8. Solid State Relay Output – Status 27
4.9. Digital Input - Status Recording 27

5. CONFIGURATION 29
5.1. General Information about Configuration Software ICP 100 29
5.2. Setting Address and Baud Rate of an e.bloxx 29
5.3. e.bloxx Settings 30
5.4. Definition of Channels 31

6. SPECIFICATIONS 33
6.1. Analog Inputs 33
6.2. Analog/Digital Conversion 34
6.3. Digital In- and Output 34
6.4. Communication Interface 34
6.5. Power Supply 34
6.6. Mechanical 35
6.7. Connection 35
6.8. Environmental Conditions 35

7. DECLERATION OF CONFORMITY 36

1. ABOUT THIS MANUAL

This manual describes the installation and setup of the e.bloxx A1-1, A1-4 and A1-8 modules. Those modules only differ in their amount of analog and digital inputs and outputs. In this manual the e.bloxx A1-1 is described and shown in the pictures and at every point where there are differences between the e.bloxx A1-1 and the other modules there will be a special note.

The following information can be found in this manual:

- Description of the e.bloxx system with detailed information on the hardware and module features.
- Installation description of the modules and how they are connected to the power supply and bus lines.
- Description of the different types of measurement.
- A short introduction on how the e.bloxx modules are configured with the CONFIGURATION SOFTWARE ICP 100. This software has an integrated help including a detailed description of the configuration process.
- Possible errors and its solutions.
- Technical specifications of the modules.

2. MODULE DESCRIPTION

2.1. System Overview

The e.bloxx modules have been developed for the industrial and experimental testing technology, especially for the multi-channel measurement of electrical signals of thermal or mechanical data at test beds and test sites.

Picture 2.1 - e.bloxx A1-1, A1-4 and A1-8

The e.bloxx A1-1, A1-4 and A1-8, which are described in this manual, are 1-, 4- or 8-channel modules for measurements of almost any kind of signals. They are part of a whole product line of different e.bloxx and differing by their number of inputs (analog or digital) and outputs. With each analog channel 7 additional channels are available for calculations, alarms, setpoints and digital signals.

Due to the fast and precise signal conditioning the e.bloxx modules produce reliable and exact measurement data.

Standardized interfaces guarantee the integration of up to 127 modules into a single network.

With the e.gate module very high data rates via Profibus-DP and Ethernet can be realized. The customer-specific signal processing supplements the standard conditioning of the single e.bloxx modules.

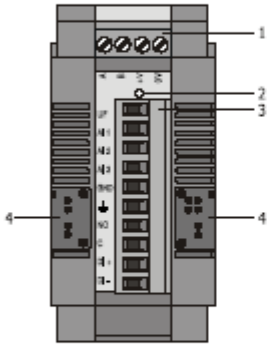
2.2. Types of Modules

There are several types of e.bloxx, which differ in their number and type of analog and digital inputs and outputs.

	A1-1	A1-4	A1-8	A3-1	A3-4	A4-1	A4-4	A5-1	A5-1 CR	A6-1	A6-3	A6-2	A9-1	D1-1	D1-4	D2-1
Voltage Supply	10 - 30 VDC															
Power Consumption [W]	1,5	6	12	1,5	6	1,5	6	1,5	1,5	2	6	5	2,3	1,5	6	2
Variable / Channels	8	32	64	8	32	8	32	8	8	8	24	16	8	8	32	8
Analog Inputs	1	4	8	4	16	4	16	2/36	2	1	3	1	-	-	-	-
Analog Outputs	-	-	-	-	-	-	-	-	-	1	3	2	4	-	-	-
Relay Outputs	1	4	8	-	-	-	-	-	-	-	-	-	-	-	-	4
Digital Inputs	1	4	8	1	4	-	-	1	1	1	3	6	1	8	32	-
Digital Outputs	-	-	-	1	4	-	-	1	1	1	3	4	1	8	32	-
Fieldbus Interface	RS 485															
Protocols	ASCII - Modbus-RTU - Profibus-DP - LocalBus															
Quantity to measure / Sensor Principle																
Voltage	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	-
Current	x	x	x	-	-	-	-	-	-	-	-	-	-	-	-	-
Resistance	x	x	x	-	-	-	-	-	-	-	-	-	-	-	-	-
Pt100 / Pt1000	x	x	x	-	-	-	-	x	-	-	-	-	-	-	-	-
Cryo Sensor	-	-	-	-	-	-	-	-	x	-	-	-	-	-	-	-
Thermocouple	x	x	x	-	-	x	x	-	-	-	-	-	-	-	-	-
Strain Gauge Full Bridge	x	x	x	-	-	-	-	-	-	x	x	x	-	-	-	-
Strain Gauge Half Bridge	-	-	-	-	-	-	-	-	-	x	x	x	-	-	-	-
Strain Gauge Quarter Br.	-	-	-	-	-	-	-	-	-	x	x	x	-	-	-	-
Inductive Full Brigde	-	-	-	-	-	-	-	-	-	x	x	x	-	-	-	-
Inductive Half Bridge	-	-	-	-	-	-	-	-	-	x	x	x	-	-	-	-
LVDT	-	-	-	-	-	-	-	-	-	x	x	x	-	-	-	-
Potentiom. Transducer	x	x	x	-	-	-	-	-	-	-	-	x	-	-	-	-
Piezoresist. Transducer	-	-	-	-	-	-	-	-	-	-	-	x	-	-	-	-
Status	x	x	x	x	x	-	-	x	x	x	x	x	x	x	x	-
Frequency	-	-	-	-	-	-	-	-	-	-	-	x	-	x	x	-
Counter	-	-	-	-	-	-	-	-	-	-	-	x	-	x	x	-

Table 2.1. - Characteristics of the e.bloxx modules

2.3. Module Parts



1 ... Pluggable Screw-Type Terminal Strip for Connection of RS-485 Bus and Power Supply
2 ... Power/Error-LED (red/green)
3 ... Pluggable Screw-Type Terminal Strip for Sensor Connection
4 ... Rapid Bus Link Plugs

Picture 2.2 - Parts of the e-blocx A1-1

Terminal Strip for RS 485 and Power Supply

Terminal	Description
A	RS 485 Bus Interface A
B	RS 485 Bus Interface B
+V	Power Supply +
0V	Power Supply -

Table 2.2 - Description of Terminal Strip for RS 485 Bus and Power Supply

Terminal Strip for Sensor Connection

Terminal	Description
UF	Force Output to Supply Measurement Voltage
AI 1	Analog Input 1
AI 2	Analog Input 2
AI 3	Analog Input 3
GND	Analog Ground
⏚	Grounding
NO	Solid State Relay Output - Normally Open
C	Solid State Relay Output - Common
DI +	Digital Input +
DI -	Digital Input -

Table 2.3 - Description of Terminal Strip for Sensor Connection

2.4. Functional Overview

This manual describes the e.bloxx modules A1-1, A1-4 and A1-8. These modules are all 8-channel modules (real plus virtual). They differ only in the number and kind of inputs and outputs. The e.bloxx A1-1 has one analog input, one digital input and one solid state relay output (opto MOSFET). The e.bloxx A1-4 has four of each and the e.bloxx A1-8 eight and all of them can be configured separately.

Each e.bloxx module has 8 channels that can be defined. The first channel always defines the type of measurement. The other channels can be used to output a value at the relay output, to process the digital input which can be used for example as a trigger, to make arithmetical calculations or to process the measured value (scaling, min/max, alarm). The channels are defined in the configuration table of the Configuration Software ICP 100.

The values of each channel can be read out via the RS 485 bus. It is possible to set a filter frequency of up to 1 kHz for the analog input. It determines how fast the measured value will be updated. The resulting update time depends on the type of measurement, for example for a resistance measurement in 3-wire technology 2 measurements are required which means that the update time is decreased accordingly. The measuring rate is specified in the part Technical Specification.

2.5. Front-LED

The LED at the front of the e.bloxx modules provides the following information:

LED green	Module works well, no signal overflow, no communication error...
LED red	general error like signal overflow, broken sense leads
LED red + short off period	general error like signal overflow, broken sense leads + communication timeout
LED green + short red flash	Signal ok + communication timeout
LED red fast flashing	global error, no suitable firmware

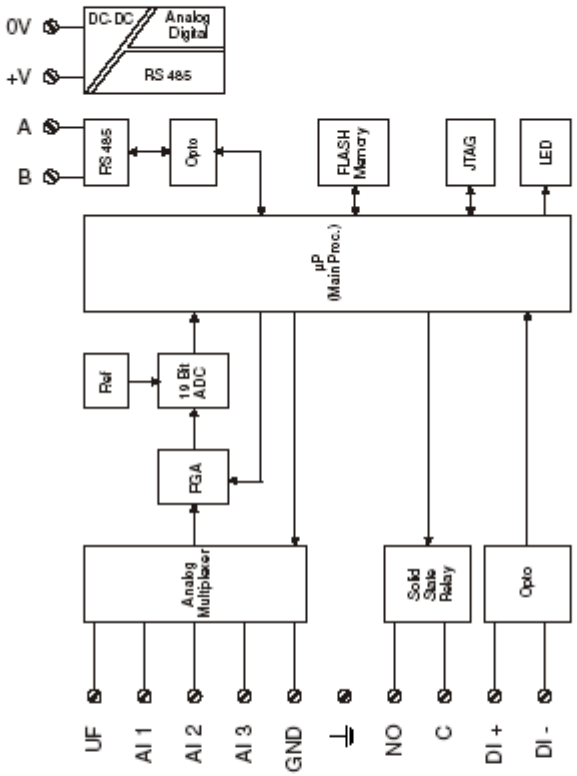
Notice: The LED will get red when the signal leaves the selected range and the error checking is activated (see ICP 100 column Range/Error).

2.6. DC-Isolation

The power supply, bus interface, analog inputs, digital inputs outputs are DC-isolated from each other.

2.7. Functional Diagram

The following picture can describe the e.bloxx A1-1.



Picture 2.3 - Functional Diagram of the e.bloxx A1-1

3. MOUNTING e.bloxx AND CONNECTING WIRES

3.1. Environmental Conditions

The e.bloxx modules are protected against water and dirt according IP 20. If required by the conditions of the operating site the modules have to be installed accordingly, e.g. in a water-resistant or water-proof case, compliant with the regulations of electrical engineering.
For the allowed ambient temperatures for the e.bloxx A1-1, e.bloxx A1-4 and e.bloxx A1-8 see the Technical Specifications at the end of this manual.

3.2. Connection Technique

The wires are connected to the modules via screw-type terminals. The captive terminal screws are part of the terminal strips. All terminal strips are of plug-in type and can be detached from the modules.

Not more than 2 leads should be connected with one clamp. In this case both leads should have the same conductor cross-section. For the precise clamping of stranded wire we recommend the use of wire-end ferrules.

Notice: Connecting wires respectively the plugging-in and -out of the terminal strip is only allowed with modules in power-off status.
In order to prevent interference with sensors, signals and modules, shielded cables have to be used for the power supply, bus connection and signal lines.
We strongly recommend using a single screened cable each input signal. To use more signals in one cable could generate interacting influences.

Notice: For optimal performance the e.bloxx modules must be grounded properly. This is achieved by utilizing the Ground/Earth screw on the back of each e.bloxx module. The screen of the sensor cable has to be grounded at the same potential.

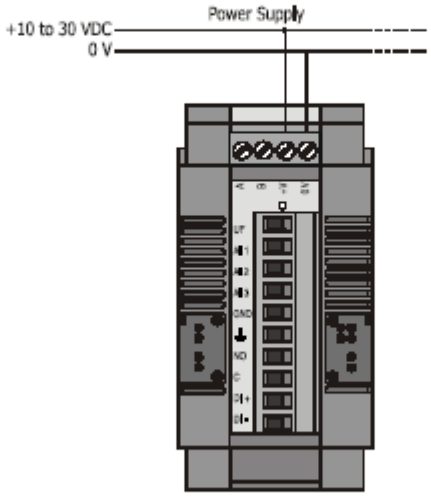
3.3. Power Supply

Non-regulated DC voltage between +10 and +30 VDC is sufficient for the power supply of the modules. The input is protected against excess voltage and polarity connecting error. The power consumption remains approximately constant over the total voltage range, due to the integrated switching regulator.

Due to their low current consumption the modules can also be remotely supplied via longer lines. Several modules can be supplied in parallel within the permissible voltage range and drop in the lines. If required, the supply lines together with the bus line may be incorporated in one cable.

In order not to overload the module power supply needlessly and to avoid unnecessary line troubles, a separate power supply is recommended for sensors with a large current drain.

The distribution voltage for the e.bloxx modules has to be protected by a fuse with maximum 1 A (inert). The modules have an internal fuse (reversible) for protection against excess voltage, excess current and wrong polarity.



Picture 3.1. - Power Supply of the e.bloxx Modules

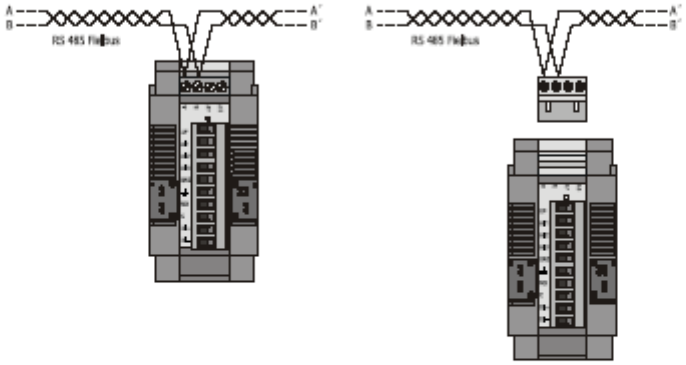
3.4. Bus Connection

Only the connection of the e.bloxx modules to the bus is described in here. A detailed description of the bus and the communication of the modules can be found in the Communication Guide of the e.bloxx/e.gate modules.

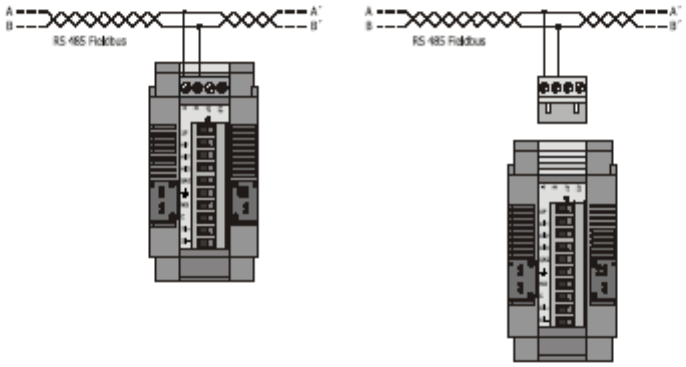
The e.bloxx modules have an RS 485 bus interface for connection to the serial fieldbus. The bus has to be terminated on both sides with a characteristic impedance. The maximum line length depends on the transmission speed (refer to the Communication Guide for details) and can never be higher than 1.2 km per bus segment or 4.8 km via a physical bus string by using 3 repeaters. A maximum of 32 devices are possible with each bus segment and up to 127 devices via a physical bus string.

Wiring

In general, the e.bloxx is connected to the bus by connecting both signal leads A and B of the incoming bus cable and A' and B' of the outgoing bus cable together to one terminal on the module (Picture 3.2). Alternatively, the bus can also be connected by a "stub cable" (Picture 3.3). This guarantees that the bus connection to other modules remains in place, even if one module has to be exchanged, due to the removable terminal strip.



Picture 3.2 - Bus Connection of an e.bloxx A1-1 to the RS 485 Fieldbus with Derivation



Picture 3.3 - Bus Connection of an e.bloxx A1-1 to the RS 485 Fieldbus via Stub Cable

The stub-cable should be as short as possible, not longer than 30 cm.

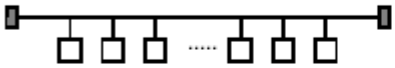
Notice: The terminal designations A and B of all modules of the e.bloxx series are exchanged compared with the PROFIBUS-definitions. Consequently, in multi-vendor systems the bus lines A and B have to be exchanged when connecting them to the e.bloxx.

Bus Structure

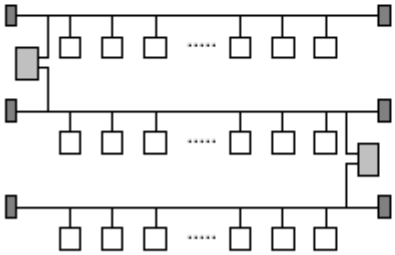
The bus structure is a line structure where each bus segment will be terminated with characteristic impedance on both ends. Branches can be set up by means of a bi-directional signal amplifier, so-called repeaters. Other types of branches are not permitted (no tree topology). The max. stub-length to a user may not exceed 30 cm.

The following figures show a few examples of possible bus topology structures. The meanings of the symbols are as follows:

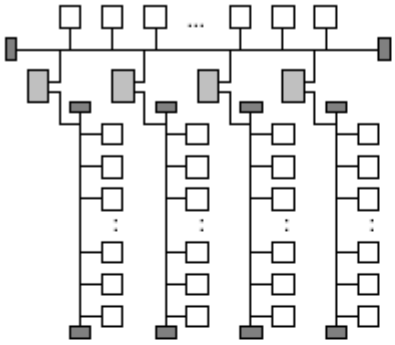
... Bus User
... Repeater
... Bus Termination



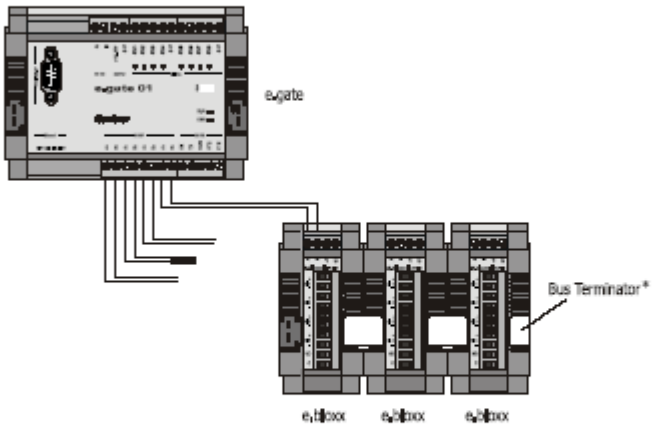
Picture 3.4. - Simple Line Structure



Picture 3.5. – Extended Line Structure



Picture 3.6. - Line Structure with Branches



Picture 3.7. - e.bloxx A1-1, A1-4 and A1-8 connected to e.gate

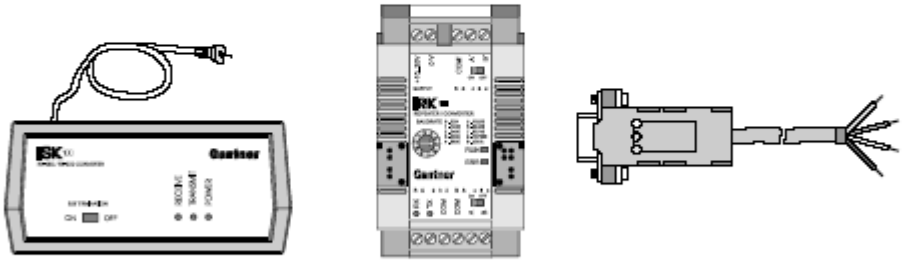
* ... If the e.bloxx A1-1, A1-4 and A1-8 are used together with an e.gate, which is used to collect the data of all connected e.bloxx modules and processes them for fast transmission via the further network, a bus termination must be connected to the last e.bloxx in each bus line.

Bus Connection to PC

The bus interface of the e.bloxx is based on the RS 485 standard. Since most hosts are "only" equipped with RS-232 interfaces, an interface converter or a plug-in board with RS 485 drivers is required for conversion purposes.

Gantner Instruments Test & Measurement GMBH offers a compact interface converter, called ISK 200, with an integrated power supply and automatic baud rate detection. The power supply, bus connection and a separate 24 VDC-output are DC-isolated. Therefore, the interface converter ISK 200 is also applicable as a power supply for remote applications. Additionally, the interface converter ISK 200 features the option of connecting the required bus termination via a switch. The converter is designed to be used as a desk device.

Another module IRK 100 from Gantner Instruments Test & Measurement GMBH is available which may be used as an RS 485 repeater or RS 485/RS 232 converter. The baud rate can be adjusted at the IRK 100. Also, for this module the required bus termination may be connected with a switch. The Repeater/ Converter IRK 100 has a snap-on mounting mechanism for the installation on standard profile rails (DIN rail) 35 mm according to DIN EN 50022.



Interface Converter ISK 200 | Repeater/Converter IRK 100 | Interface Converter ISK 101

Picture 3.8. - Interface Converters ISK 200, IRK 100 and ISK 101

Bus Connection to Profibus-DP

For the installation of the bus cable and bus interface, 9-channel D-subminiature plugs and sockets are used. The pin assignment for the RS 485 connection according to PROFIBUS is shown in Table 3.1.

Plug	Pin	RS 485 Notation	Signal	Identification
 DB 9	1	-	Shield	Shield, Protective Ground
	2	-	RP	Reserved for Power
	3	B / B'	RxD/TxD-P	Receive/Transmit-Data-P
	4	-	CNTR-P	Control-P
	5	C / C'	DGND	Data Ground
	6	-	VP	Voltage Plus
	7	-	RP	Reserved for Power
	8	A / A'	RxD/TxD-N	Receive/Transmit-Data-N
	9	-	CNTR-N	Control-N

Table 3.1. - Pin Assignment D-Subminiature Plug According to PROFIBUS

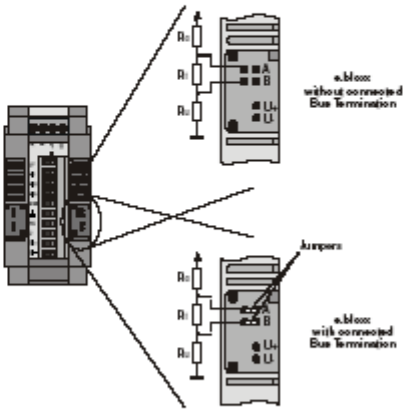
The signal leads A and B (and Shield) are mandatory for a (shielded) connection. Additional signal leads may be installed if required.

Notice: Due to the fact that the RS 485 interface is used for different protocols, in case of using Profibus-DP the leads A and B has to be crossed.

Bus Termination at the e.bloxx Modules

In order to avoid signal reflections on the bus, each bus segment has to be terminated at its physical beginning and at its end with the characteristic impedance. A terminating resistor is installed between the bus leads A and B for this purpose. In addition, the bus lead A is connected via a pull-up resistor to potential (VP) and the bus lead B is connected via a pull-down resistor to ground (DataGround). These resistors provide a defined quiescent potential in case there is no data transmission on the bus. This quiescent potential is level high.

The e.bloxx modules have built in these bus termination resistors. They can be connect to the bus by plugging the Bus *Termination Plug IBT 100*, which is available as accessory, into the rapid bus link plug on the front side of the module. Instead of the bus termination plug *IBT 100*, also separate jumpers may be used for the bus termination. In this case, it is mandatory that the jumper clips are installed as indicated below, and that the bus leads or the bus termination are not short-circuited by mistake.



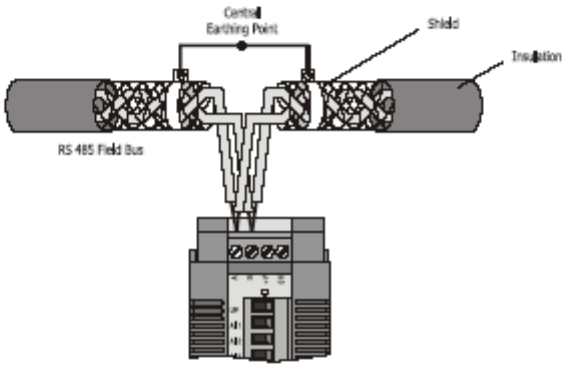
Picture 3.9. - Bus Termination at the e.bloxx Modules

3.5. Shielding

In case of increased interference, such as in industrial areas, we recommend shielding of bus and signal cables. In general, the shield should be connected to the protective grounding (not DataGround!) at each bus connection. If necessary, the shield should also be applied along the course of the cable several times. For shorter distances, e.g. with stub cables, the interference response is often improved if the shielding is only applied to the stub cable exit.

Bus users such as controllers (PLCs), computers (PCs), repeaters and interface converters (ISK), etc., generally feature the possibility of applying the shield directly to the appliance or to separate shield rails. Shield rails offer the advantage of preventing possible interfering signals from reaching the appliance. The shields, which are connected to protective grounding, conduct interference signals off before reaching the module.

The e.bloxx do not have a direct shield connection at the module. Here the shield of the bus cable can be connected to earth e.g. by so-called shield clamps.



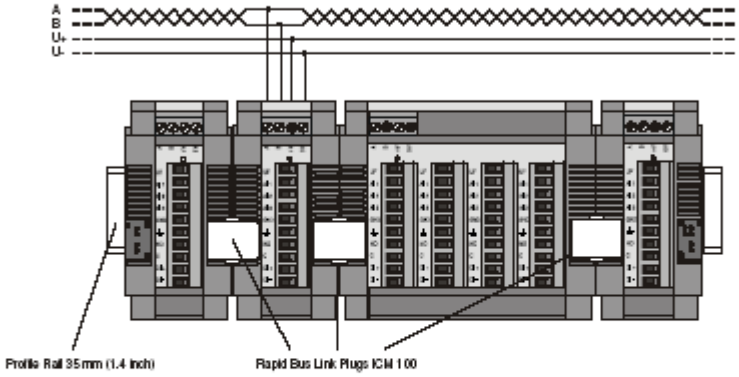
Picture 3.10. - Grounding of the Bus Line Shield at an e.bloxx

Notice: The shielding screen must not be connected to the ground (0V) of the power supply and it should always be connected to earth with a large surface and low-inductance.

3.6. Rapid Bus Link Plug

The e.bloxx have plugs on the left and right side, which allow connecting the bus and power supply from one module to the next with a Rapid Bus Link Plug (type designation: *ICM 100*). This kind of connecting bus and power supply is particularly advantageous if several modules are mounted on one common profile rail side by side. In this case, only the terminal of one module has to be connected. Furthermore, various modules of the e.bloxx series may be connected with the Rapid Bus Link Plug.

Notice: The current flowing through the Rapid Bus Link Plug Jack and the e.bloxx must not exceed 1 A. Thus, the power supply should preferably be connected to the middle of several modules and no more than 6 pieces of e.bloxx may be connected via the Rapid Bus Link Plug *ICM 100* in one line.



Picture 3.11. - Connection of four e.bloxx Modules with Rapid Bus Link Plugs ICM 100

4. MEASUREMENTS

4.1. General

The e.bloxx A1-1, A1-4 and A1-8 have 1, 4 and 8 analog input(s), relay output(s) and digital input(s). Depending on the type of sensor, which is connected to the analog input(s), various numbers of terminals have to be used. The configuration of the inputs and outputs is done with the Configuration Software ICP 100 as required by the application. For the e.bloxx A1-4 and A1-8 the 4 resp. 8 physical in-/outputs can be configured independently from each other.

Analog Input

The analog input collects and processes the signals of the most common transducers. Currently data of a large number of standardized and proprietary sensors are stored in the e.bloxx. The user can input further sensor data. The acquisition of various measuring values with these sensors may be reduced to a few principles of measurement, which are:

- Measurement of Voltage
- Measurement of Current
- Measurement of Resistance incl. Pt100 and Pt1000
- Measuring with a Resistance Bridge
- Measurement of Temperature with Thermocouples

For some of these measurements the e.bloxx offers several types of measurement. Currents up to ±24 mA may directly be measured by the e.bloxx. Measuring the voltage drop at an external shunt can carry out measurements of currents over ±24 mA. Resistance measurements can be carried out in 2-, 3- and 4-wire technique, or with resistance bridges in 4-wire technique. When measuring temperature with thermocouples, the user can choose between internal and external cold junction compensation.

The analog input is protected against excess voltages.

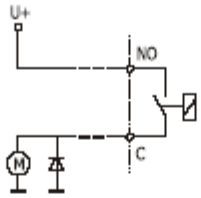
Notice: Overloads of more than ±10 VDC will result in false measurement data at the analog input channel.

Solid State Relay Output (Opto MOSFET)

The relay output of the e.bloxx can be configured as threshold switch with definable threshold-settings and may then be used e.g. as an alarm or limit monitor of the analog input.

Since the relay output is "passive" the process of external elements always requires an external current supply. In case of larger loads this current supply should be independent of the module supply. When connecting inductive loads a freewheeling diode is recommended in order to prevent possible disturbances e.g. by induced voltage (see picture 4.1).

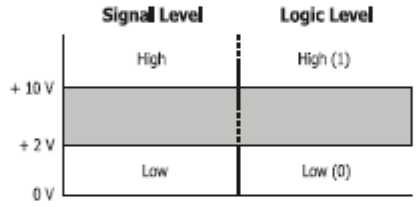
To the relay output you can directly connect: signal lamps, small relays, switching relays for larger loads, acoustic signal installations, buzzers or beepers etc., as long as the connected loads do not exceed the values described in the Technical Specifications.



Picture 4.1. – Free-Wheeling Diode at Relay Output

Digital Input

The input of the e.blocx can be used for collecting status information. The maximum permissible input voltage amounts to 30 V. Input voltages between 10 VDC and 30 VDC are interpreted as logic HIGH ("1"), input voltages lower than 2 V as logic LOW ("0"). The maximum fan-in current amounts to 5 mA.

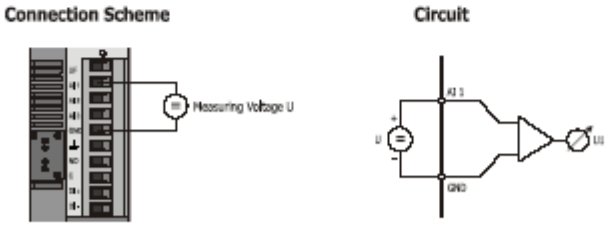


Picture 4.2 - Definition of Signal Levels and Logic Levels

Internal Reference Voltage

An internal reference voltage serves to balance the entire analog signal processing automatically.

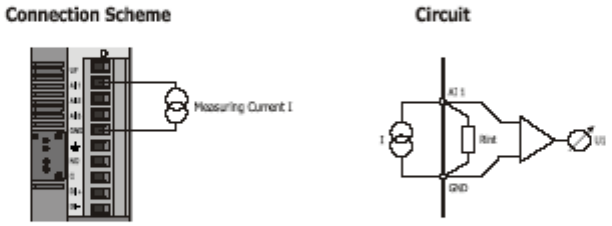
4.2 Analog Input - Measurement of Voltage



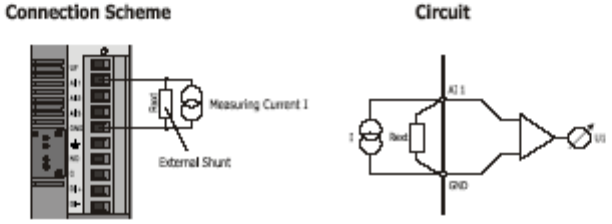
Picture 4.3 - Measurement of Voltage - Single-Ended

With the single-ended type of measurement the voltage to be measured is connected between an analog input (AI 1 .. 3) and analog ground (GND). The measurement voltage may not exceed 10 VDC.

4.3. Analog Input - Measurement of Current



Picture 4.4. - Measurement of Current with Internal Shunt



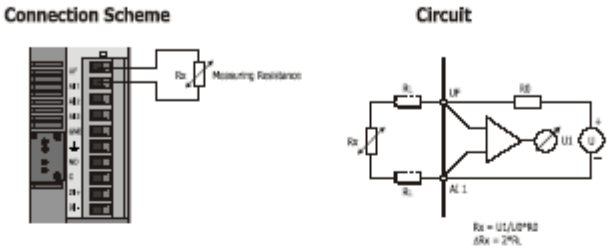
Picture 4.5. - Measurement of Current with External Shunt

For measurements of current with the e.bloxx the source of electricity is connected to an analog input AI 1 and the analog ground GND. For the measurement, the required load on the current source is regulated by an internal resistor R_{int} with a value of 100 Ω. The maximum power of this shunt is limited to 0.25 W, resulting in a measuring range of up to 25 mA maximum.

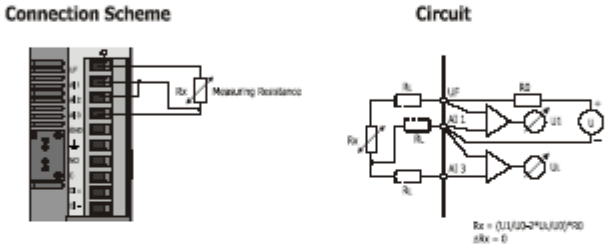
If higher currents need to be measured, an external resistor that is connected parallel to the source of current should be used. Terminals are connected to the analog voltage input AI 1 and analog ground GND. The power of the external shunt has to be adapted to the source of current to be measured in order to limit the voltage at the analog input to ±10 V. The analog input is configured as voltage input. The voltage has to be divided by R_{ext}.

Notice: The precision of the current measurement with external shunt depends on the accuracy of the resistor that is used.

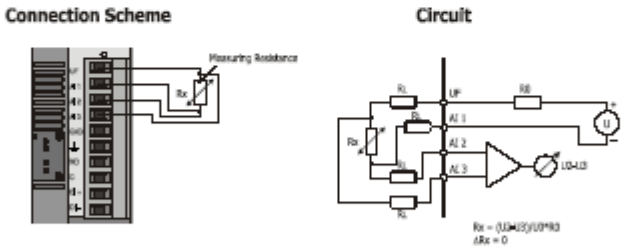
4.4. Analog Input - Measurement of Resistance and RTD (Pt100, Pt1000)



Picture 4.6. - Measurement of Resistance in 2-Wire Technique



Picture 4.7. - Measurement of Resistance in 3-Wire Technique



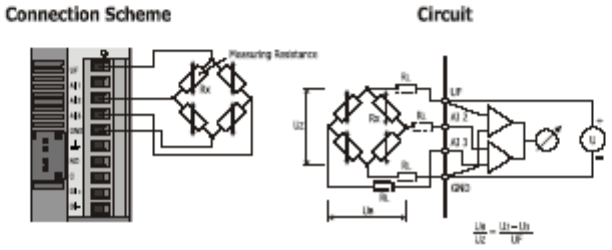
Picture 4.8. - Measurement of Resistance in 4-Wire Technique

Resistance measurement is carried out by means of measurements of voltages at a current-carrying resistor. In this case the occurring voltage drop is measured via the resistance sensor. The current feed required for the resistance measurements provides the internal supply of the module.

For this purpose the sensor module connects a supply point internally with the analog measurement input via a reference resistor Ro. The drop of voltage Uo via the resistor Ro is required as a reference for further signal processing by the module. The value of resistance of the sensor can be calculated from the input signals Ui as a multiple of the reference resistor Ro. The measuring range amounts to between 0 and 4 kΩ.

Notice: The e.bloxx support resistance measurement in 2-, 3- and 4-wire technique. With resistance measurement in 2-wire technique the supply lines cause an additional drop of voltage, thus distorting the measuring result and influencing the measuring accuracy. Therefore it is necessary to pay attention especially with resistance measurement in 2-wire-technique to use as low-impedance leads as possible to the sensors and to make sure that the leads are well-connected with the sensor module and the sensor. With resistance measurement in 3-wire technique the drop of voltage via the supply lines is eliminated from the measuring result (-2·UL/U0·R0). In this case 2 measurements are required (for U1/U0 and UL/U0). With resistance measurement in 4-wire technique the drop of voltage is picked up directly at the sensor, so that the supply lines do not influence the measuring results any longer. The measuring frequency for these resistance and Pt100 measurements is 10 Hz

4.5. Analog Input - Measurement with a Resistance Bridge



Picture 4.9 - Measurement by a Resistance Bridge

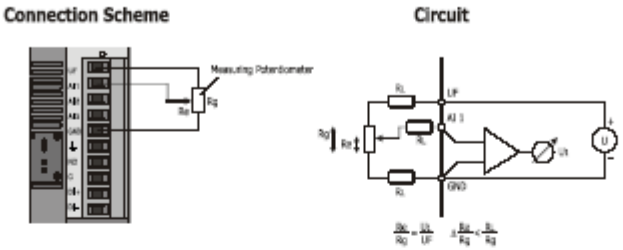
Bridge connections consist of 2 arms with two resistors each. The resistance bridge is supplied by the voltage output UF at the e.bloxx.

The quantity to be measured with resistance bridges is the relation between bridge voltage UZ and the voltage between the two resistance arms UB (ratio measurement). The measuring ranges are ±2 mV/V, ±8 mV/V, 20 mV/V, 200 mV/V and 1000 mV/V.

Mostly there are four variable resistors (e.g. strain gauges) in resistance bridges, so that the resistance bridge can easily be balanced via the controllable resistor (UB = 0 for the balanced state). Variations of the sensor signal characteristically influence the fourth resistor and cause a change in the quantity to be measured.

Notice: The supply lines cause an additional voltage drop, which leads to a slight diminishing of the bridge voltage. This distorts the measuring result and thus influences the measuring accuracy. Therefore it is necessary to pay attention to use as low-impedance leads as possible to the sensors and to make sure that the leads are well connected with the e.bloxx and the sensor.

4.6. Analog Input - Potentiometer Measurement



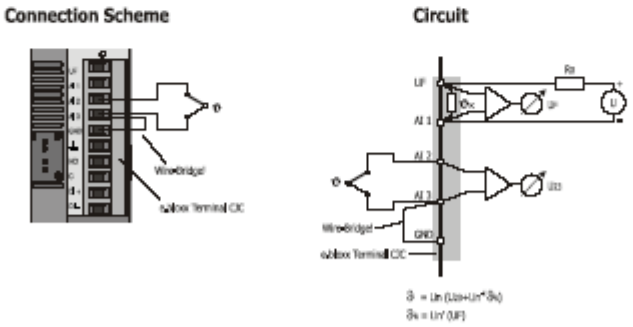
Picture 4.10. - Potentiometer Measurement

Potentiometer measurements are measurements with voltage relations, the division ratio of which can be adjusted (e.g. by a sliding contact on a resistance regulator). The quantity to be measured is the relation between the adjusted resistance RE and the combined resistance RG of such a potentiometer (ratio measurement).

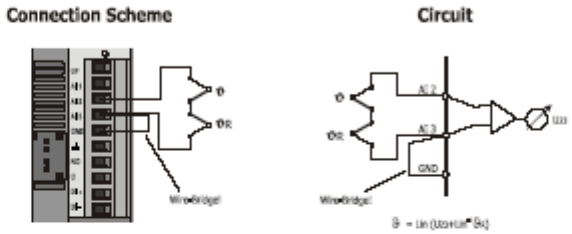
With the e.bloxx the potentiometer is supplied by the voltage output UF on the e.bloxx. The signal is picked up at the resistor.

Notice: With potentiometer measurements the supply lines cause an additional voltage drop, which can lead to a slight decrease in signal voltage, thus distorting the measuring result and influencing the measuring accuracy. Therefore it is necessary to pay attention with potentiometer measurement to use as low-impedance leads as possible to the sensors and to make sure that the leads are well connected with the e.bloxx and the sensor.

4.7. Analog Input - Measurement of Temperature with Thermocouples



Picture 4.11. - Measurement of Temperature with Thermocouple - Internal Cold Junction Compensation with the e.bloxx Terminal CJC



Picture 4.12 - Measurement of Temperature with Thermocouple - External Cold Junction Compensation

Thermocouples consist of two "thermoelectric wires" made of different materials (e.g. platinum and platinum rhodium) that are welded to each other at one end. If the contact position and the other ends of the thermoelectric wires have different temperatures, a "thermoelectric voltage" U_{th} appears at the contact position of both thermoelectric wires. This voltage is largely proportional to the temperature difference. It can be measured and can be used for temperature measurement purposes.

Since thermocouples can only measure a temperature difference (difference between temperature to be measured and temperature at the connecting terminals on the e.bloxx), a terminal temperature or a known temperature reference also have to be determined. In the first case this is called internal cold junction compensation (TC_{int}), in the second case external cold junction compensation (TC_{ext}).

When measuring the temperature with internal cold junction compensation a temperature sensor will be connected at an additional analog input next to the thermocouple. Or by means of Cold Junction Terminal – e.bloxx Terminal CJC - where a Pt1000 temperature sensor is integrated directly in the terminal block between the terminal connections UF and AI 1, the temperature ϑk will be entered. The temperature of the test point is determined by the e.bloxx because of linearization trace to

$$\vartheta x = \mathrm{Lin}(Ux + \mathrm{Lin}^{-1}\vartheta k), \text{ where } \vartheta k = \mathrm{Lin}'(UF).$$

The sensor module will be informed about the measuring channel through which the temperature of terminals can be calculated via the Configuration Software ICP 100 (Cold Junction Channel).

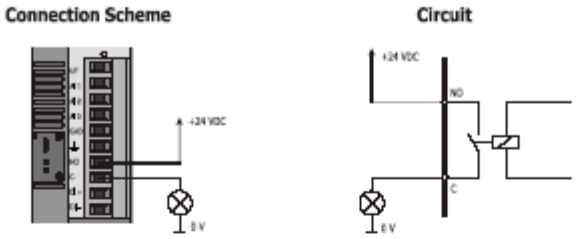
If the temperature is measured by external cold junction compensation, a second thermocouple of the same type is required, which is connected in series with the first one. The polarity is selected in a way that the thermoelectric voltages subtract each other. The second thermocouple is set to a fixed reference temperature ϑr (mainly ϑr = 0 °C). Then the e.bloxx calculates the temperature at the measuring position by means of the linearization curve as

$$\vartheta x = \mathrm{Lin}(Ux + \mathrm{Lin}^{-1}\vartheta r).$$

The e.bloxx will be informed about the reference temperature ϑr via the Configuration Software ICP 100 (Cold Junction Temperature).

A wire bridge has to be placed between the GND and one of the thermocouple inputs (see pictures 4.11 and 4.12).

4.8. Solid State Relay Output – Status



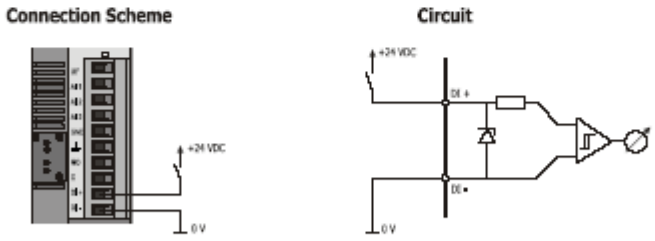
Picture 4.13. - Relay Output used for Status Output

The relay output can be used to output *host-controlled* or *process-controlled* status signals.
With the host-controlled digital status output, the digital output is set according to the status information received via bus. With the process-controlled output of status information the e.blocx monitors measured values, resp. sensor variables as to certain threshold values. The digital output is set if one or several threshold conditions are fulfilled. The user via the Configuration Software ICP 100 can freely define the threshold values. The user can also preset the logical signal level.

The distribution voltage can amount from 10 up to 30 VDC. It has to be supplied externally or picked up by the power supply of the e.blocx.

The status of the digital output can be scanned as 1/0 information via bus.

4.9. Digital Input - Status Recording



Picture 4.16 - Digital Status Recording

Picture 4.17 - Signal Diagram of Status Recording

For the acquisition of digital status information (on/off, closed/open, left/right, etc.) the signal applied to the digital input is collected and is held ready for further processing in the e.bloxx or for transmission via bus.
The digital input is set (switch closed) as long as the applied signal voltage remains over the threshold value of 10 V.
The digital information can be scanned as 1/0 information via bus.

5. CONFIGURATION

5.1. General Information about Configuration Software ICP 100

The e.bloxx modules can be configured with the Configuration Software ICP 100. This software includes all functions to set the module parameters like baud rate, measurement rate, etc. and to define the input and output functions like the type of measurements and the processing of the measured values.
The Configuration Software ICP 100 also includes a function to display measured values in real-time. There are also several software packages from other companies that are adapted to the specific measurement tasks.
In the Configuration Software ICP 100 the two register cards "Variable Settings" and "Module Settings" will be displayed if you are configuring an e.bloxx, which is not online, and also the two additional register cards "Info" and "Measure" when configuring an online e.bloxx.

- On the register card "Info" several module information will be displayed.
- On the register card "Measure" the channel values of the online e.bloxx will be displayed in real time.
- On the register card "Variable Settings" the different channels of the e.bloxx can be configured. This will be done in the Variable Settings Table being displayed on this register card.
- On the register card "Module Settings" different general settings like the baud rate, address, etc. can be defined for each e.bloxx.

This manual only gives a brief description on how to set up and configure an e.bloxx module. A detailed description of all the functions of the Configuration Software ICP 100 is included in the help function of the software.

5.2. Setting Address and Baud Rate of an e.bloxx

Before a control (PLC) or a computer (PC) can interchange data with an e.bloxx via the bus, address and baud rate of the e.bloxx have to be defined. The following points have to be taken into consideration in this connection:
- All devices have to be adjusted to the same baud rate.
- The same address must not appear twice in the bus topology.

The setting variants for the bus parameters for e.bloxx are:

Bus Parameter	ASCII Protocol	MODBUS Protocol	Profibus-DP Protocol	LOCAL-BUS Protocol
Address	1 127	1 127	1.....126	1 127
Baud Rate	19,200 bps	19,200 bps	19,200 bps	19,200 bps
	38,400 bps	38,400 bps	-	38,400 bps
	57,600 bps	57,600 bps	-	57,600 bps
	93,750 bps	93,750 bps	93,750 bps	93,750 bps
	115,200 bps	115,200 bps	-	115,200 bps
	-	-	187,500 bps	187,500 bps
	-	-	500,000 bps	500 kbps
	-	-	1,500 kbps	1,500 kbps

Table 5.1 - Setting variants for address and baud rate for the e.bloxx

If no other specifications are made on delivery, the e.bloxx have address 1 and baud rate 1.5 Mbps as default. The adjustment can be changed via the bus by means of the *Configuration Software ICP 100*.

Adjustment via bus by means of the Configuration Software ICP 100:
The address and baud rate of an e.bloxx can be set in the Configuration Software ICP 100. On the dialog box "Module Information" the address and baud rate of the actual e.bloxx is displayed. After changing these settings, the new settings have to be loaded into the e.bloxx in order to take effect. To do this the menu item **Send to Module** or **Send to Module as...** in the menu **File** or the corresponding button ( ) in the icon bar has to be selected.

Notice: The address 0 is provided for the PC in case of a transmission via PROFIBUS-DP. This address therefore cannot be assigned to the e.bloxx. Also the address 127 is reserved for broadcast transmission in the PROFIBUS-DP protocol and may only be assigned for these cases.

5.3. e.bloxx Settings

On the register card "Module Settings" the following settings of an e.bloxx can be defined.

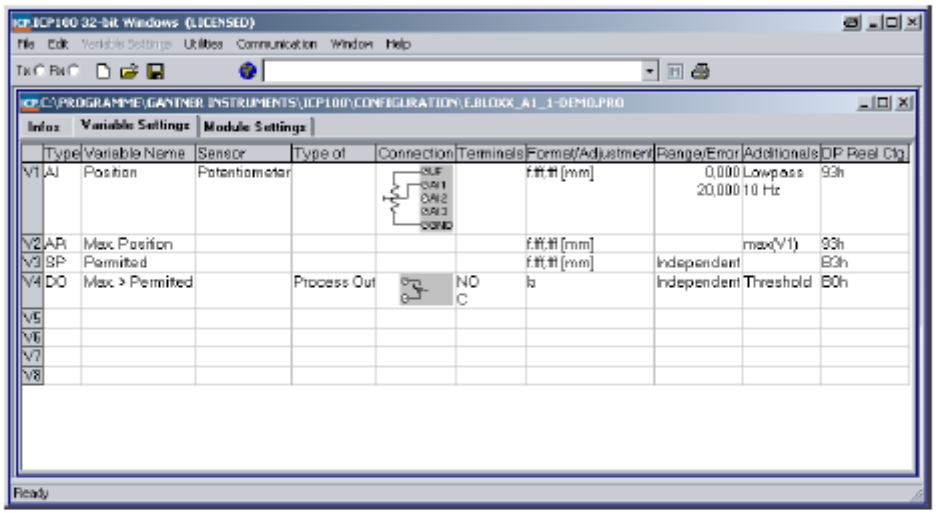
- Location: Description of each e.bloxx.
- User Name: Possibility to enter the name of the person that has configured the module.
- Config. Date: Displays the date of configuration.
- Address: Address of the online e.bloxx. Will only be displayed if the e.bloxx is online.
- Protocol: Bus protocol being used for communication between PC and e.bloxx. Will only be displayed if the e.bloxx is online. In the configuration software ICP 100 only the LocalBus protocol is displayed. Nevertheless all the protocols mentioned in chapter 2.2 are available and the e.bloxx uses the required protocol automatically.
- Character Format: Determines the number of data, parity and stop bits for transmission between PC and e.bloxx. Will only be displayed if e.bloxx is online. With the e.bloxx the character format is fixed to 8E1.
- Answer Delay: Determines how long an e.bloxx will wait before it sends an answer to a host request.
- Timeout: A timeout means that there is no communication with the module during the time period that is set here. All host-controlled functions (output via the Digital and Analog Output Variables and the Setpoint Variable) can be defined to pass into a safe, definable status. As soon as the communication recommences, the values are assumed again, depending on the configuration.
- Special Data: If a special program (firmware) is loaded in the e.bloxx it may need some special data that can be input here.

After changing some of these settings, the new settings have to be loaded into the corresponding e.bloxx, so they can take effect. Therefore select the menu item **Send to Module** or **Send to Module as...** in the menu **File** or the corresponding button ( ) in the icon bar.

5.4. Definition of Channels

Up to 8 channels (real plus virtual) can be defined for an e.bloxx. They define how the signals at the in- and outputs of the e.bloxx will be processed. The value of every channel can be read out via the fieldbus. The channels are defined in the Variable Settings Table in the Configuration Software ICP 100.

The Variable Settings Table is displayed on the corresponding register card in the Configuration Software ICP 100.



Picture 5.1. - Example for Variable Settings Table

Here all 8 possible channels will be listed. To define a new channel just click on a free line in the table or change the type of channel clicking on the first column *Type* in the corresponding line. In both cases a dialog box will be opened where the type of the new channel has to be selected. There are 6 different types of channels:

- Analog Input Channel: Used to measure analog sensor signals. In the column *Sensor* the type of connected sensor is selected (voltage, current, resistance, temperature) and in the next column the corresponding *Type of Measurement* is set (refer to chapter 4).
- Digital Input Channel: Used to record digital status signals. In the column *Type of Measurement* it is only possible to select the measurement function state recording (refer to chapter 4).
- Digital Output Channel: This is the relay output of the e.bloxx. Status signals can be output automatically by the e.bloxx according to values of other channels (process controlled) or it is possible to set the state of the output via bus(refer to chapter 4).
- Arithmetic Channel: With this channel it is possible to perform calculations with the actual values of other channels and with constant values. The results of the calculations are assigned to the arithmetic channel and so arithmetic channels can also be used by other arithmetic channels for calculations.
- Alarm Channel: An Alarm Channel can be used to monitor another channel and to generate an alarm message if one of up to 4 definable thresholds are exceeded. The alarm messages can be read via bus.

- Setpoint Channel: The value of this channel can be set via bus. This way it is possible to set a value via bus which can be used by an arithmetic channel for further processing, e.g. to set a factor for measurement by the user.

The settings for all defined channels will be displayed in the corresponding column of the channels. To change these settings click on the corresponding field in the Variable Settings Table.

One analog input, one digital input and one relay output could be defined.

6. SPECIFICATIONS

All following data are valid after a warm-up time of approx. 45 minutes.

6.1. Analog Inputs

Accuracy	0.01 % typical		
	0.02 % in controlled magnetically environment according EN61326: 1997, appendix B		
	0.05 % in industrial area according to EN61326: 1997, appendix A		
Repeatability	0.003 % typical (within 24 hours)		
Type of Measurement	Range	Accuracy	Resolution
Voltage	±10 V	±2 mV	40 µV
	±1 V	±0.2 mV	4 µV
	±100 mV	±20 µV	0.4 µV
	±10 mV	±10 µV	0.04 µV
Current	4-20 mA	±4 µA	80 nA
internal shunt 100Ω	±20 mA	±2 µA	80 nA
Resistance	4 kΩ	±1 Ω	0.05 Ω
(2-, 3- and 4-wire)	2 kΩ	±0.6 Ω	0.03 Ω
measuring current 1 mA DC			
Bridge	±1000 mV/V	1 mV/V	100 µV/V
Excitation 5 VDC/120 Ω	±200 mV/V	200 µV/V	10 µV/V
	±20 mV/V	20 µV/V	1 µV/V
	±8 mV/V	±8 µV/V	0.4 µV/V
	±2 mV/V	±2 µV/V	0.1 µV/V
RTD (2-, 3- and 4-wire)			
Pt100	-200 to + 850 °C	±0.5 °C	±0.1 °C
Pt100	-200 to + 250 °C	±0.2 °C	±0.01 °C
Pt1000	-200 to + 850 °C	±1 °C	±0.1 °C
Pt1000	-200 to + 140 °C	±0.3 °C	±0.01 °C
Thermocouples	Type B	better than ± 5°	
	Type E, J, K, L, T, U	better than ± 1°	
	Type N:	better than ± 2°	
	Type R, S:	better than ± 3°	
Common mode voltage	500 V permanent		
Linearity deviation:	0.01% of the final value		
Temperature influence			
on zero	1 µV / 10 °K		
on sensitivity	0.02 % / 10 °K		
Long-time drift	1 µV / 24 h; 0.1 µA per 24 h		

6.2. Analog/Digital Conversion

Resolution	19 bit
Sample Rate	1,000 samples/sec for voltage, current, potentiometer, bridge
	10 samples/sec for resistance, RTD
	4 samples/sec for thermocouples
Conversion method	Sigma Delta
Filter	Anti aliasing Bessel 4th order 200 Hz
	variable digital low pass filter 1st order
	Averaging, sliding averaging

6.3. Digital In- and Output

Input	
Function	Status
Input voltage	max. +30 VDC
Input current	max. 1,5 mA
Switching threshold	>10 VDC (high)
Switching threshold	<2.0 VDc (low)
Output	Solid State Relay output
Contact	Opto – MOSFET
Nominal load	60 VDC / 100 mA (ohmic load)
Galvanic isolation	500 V

6.4. Communication Interface

Standard	RS 485, 2-wire
Data format	8E1
Protocols	ASCII, Modbus-RTU, Profibus-DP, Local-Bus
Baud rates	
ASCII and Modbus-RTU	19.2, 38.4, 57.6, 93.75, 115.2 kbit/s
Profibus-DP	19.2, 93.75, 187.5, 500, 1500 kbit/s
Local-Bus:	19.2, 38.4, 57.6, 93.75, 115.2, 187.5, 500, 1500 kbit/s
Connectable devices	up to 32 without repeater
	up to 127 with repeater
Galvanic isolation:	500 V

6.5. Power Supply

Power supply	10 VDC to 30 VDC
	over voltage and overload protection
Power consumption	
e.blaxx A1-1	approx. 1.5 W
e.blaxx A1-4	approx. 6 W
e.blaxx A1-8	approx. 12 W

Influence of voltage:	0.001 % / V

6.6. Mechanical

Case	Aluminium and ABS
Dimensions (W x H x D) and weight	e.bloxx A1-1: 45 x 90 x 83 mm (1.8 x 3.5 x 3.3 inch), 160 g e.bloxx A1-4: 104 x 90 x 83 mm (4.1 x 3.5 x 3.3 inch), 500 g e.bloxx A1-8: 186 x 90 x 83 mm (7.3 x 3.5 x 3.3 inch), 900 g
Protective system	IP 20
Mounting:	DIN EN-Rail

6.7. Connection

Plug-In screw terminals	Wire cross-section up to 1.5 mm²
Rapid bus connector	4-pin plug in ABS-housing

6.8. Environmental Conditions

Operating Temperature	-20 °C to + 60 °C (-4 °F to + 140 °F)
Storage Temperature	-30 °C to +85 °C (-22 °F to + 185 °F)
Relative humidity	0% to 95% at +50 °C (+122 °F), non-condensing

7. DECLERATION OF CONFORMITY

Konformitätserklärung – Declaration of Conformity – Déclaration de Conformité

The undersigned, representing:

Gantner Instruments Test & Measurement GmbH
Montafonerstr. 8 – A-6780 Schruns /Austria
tel: +43/5556-73748-410 – www.gantner-instruments.com

herewith declares, that the product:

e.bloxx A1-1
Certificate Ref No: 040330WG-01

is in conformity with the following EC directive(s), including all applicable amendments:

Directives		Short Title
X	89 / 336 / EEC	EMC Directive
	99 / 5 / EEC	R&TTE Directive
	73 / 23 / EEC	Low Voltage Directive
	98 / 37 / EEC	Machinery Directive
	99 / 519 / EEC	Limitation of human exposure to electromagnetic Fields

Only "x"-marked directives are relevant for the product and for this declaration of conformity !

and that the standards and/or technical specifications referenced below have been applied:

Standards			Short Title
EMC		EN 61000-6-1 : 2001	Generic immunity standard for residential, commercial and light-industrial environments
	X	EN 61000-6-2 : 1999	Generic immunity standard for industrial environments
		EN 61000-6-3 : 2001	Generic emission standard for residential, commercial and light-industrial environments
	X	EN 61000-6-4 : 2001	Generic emission standard for industrial environments
	X	EN 61326: 1997+A1+A2	Electrical equipment for measurement, control and laboratory use – EMC requirements
R&TTE		EN 300220-1/3 : 2000	Electromagnetic compatibility for Short Range Devices (SRDs) from 25 to 1000 MHz
		EN 300330-1/2 : 2001	Electromagnetic compatibility for Short Range Devices (SRDs) from 9 kHz to 25 MHz
		EN 301489-1/3 : 2001	Electromagnetic compatibility for Short Range Devices (SRDs) from 9 kHz to 40 GHz
Safety		EN 61010 : 2001	Safety requirements for electrical equipment for measurement, control and laboratory use
		EN 60950 : 2000	Safety requirements for information technology equipment
		EN 60335 : 2002	Safety of household and similar electrical appliances
		EN 60601 : 1988	Safety requirements for medical electrical equipment
Machinery		EN 292-1/2: 1991	Safety of machinery – Basic concepts, general principles for design
		EN 954-1: 1996	Safety of machinery – Safety-related parts of control system
		EN 60204-1:1997	Safety of machinery – Electrical equipment
Human Expos.		EN 50354 : 2001	Limitation of human exposure to electromagnetic fields
		EN 50371 : 2002	Limitation of human exposure to electromagnetic fields (10MHz-300GHz) – Generic Standard

Remarks: Only "x"-marked standards are relevant for the product and for this declaration of conformity! Concerning safety aspects, the general and the product specific warning and safety instruction in the product accompanying documents must also be regarded!

This declaration is based upon the respective technical documentation held by the manufacturer.

CE

Schruns, 30th March 2004

Gantner Instruments
Test and Measurement GmbH
Montafonerstr. 8, A-6780 Schruns

Werner Ganahl, General Manager

Gantner Instruments Test & Measurement GmbH
Montafonerstr. 8 · A-6780 Schruns · Tel. +43 5556 73748 - 410 · Fax +43 5556 73748 - 419 · Firmenbuch FN 245178g · UID ATU57738723
Industriestr. 12 · D-64297 Darmstadt · Tel. +49 6151 95136 - 0 · Fax +49 6151 95136 - 26 · HRB 9169 AG Darmstadt · UID DE814913763
Geschäftsführung: Werner Ganahl, Reinhard Keiner · office@gantner-instruments.com · www.gantner-instruments.com

e.bloxx A1-1, A1-4, A1-8
DECLARATION OF CONFORMITY

Gantner instruments

Gantner instruments

Konformitätserklärung – Declaration of Conformity – Déclaration de Conformité

The undersigned, representing:

Gantner Instruments Test & Measurement GmbH
Montafonerstr. 8 – A-6780 Schruns /Austria
tel: +43/5556-73748-410 – www.gantner-instruments.com

herewith declares, that the product:

e.bloxx A1-4
Certificate Ref.No: 040330WG-02

is in conformity with the following EC directive(s), including all applicable amendments:

Directives		Short Title
X	89 / 336 / EEC	EMC Directive
	99 / 5 / EEC	R&TTE Directive
	73 / 23 / EEC	Low Voltage Directive
	98 / 37 / EEC	Machinery Directive
	99 / 519 / EEC	Limitation of human exposure to electromagnetic Fields

Only "x"-marked directives are relevant for the product and for this declaration of conformity!

and that the standards and/or technical specifications referenced below have been applied:

Standards			Short Title
EMC		EN 61000-6-1 : 2001	Generic immunity standard for residential, commercial and light-industrial environments
	X	EN 61000-6-2 : 1999	Generic immunity standard for industrial environments
		EN 61000-6-3 : 2001	Generic emission standard for residential, commercial and light-industrial environments
	X	EN 61000-6-4 : 2001	Generic emission standard for industrial environments
	X	EN 61326: 1997+A1+A2	Electrical equipment for measurement, control and laboratory use – EMC requirements
R&TTE		EN 300220-1/3 : 2000	Electromagnetic compatibility for Short Range Devices (SRDs) from 25 to 1000 MHz
		EN 300330-1/2 : 2001	Electromagnetic compatibility for Short Range Devices (SRDs) from 9 kHz to 25 MHz
		EN 301489-1/3 : 2001	Electromagnetic compatibility for Short Range Devices (SRDs) from 9 kHz to 40 GHz
Safety		EN 61010 : 2001	Safety requirements for electrical equipment for measurement, control and laboratory use
		EN 60950 : 2000	Safety requirements for information technology equipment
		EN 60335 : 2002	Safety of household and similar electrical appliances
		EN 60601 : 1988	Safety requirements for medical electrical equipment
Machinery		EN 292-1/2: 1991	Safety of machinery – Basic concepts, general principles for design
		EN 954-1: 1996	Safety of machinery – Safety-related parts of control system
		EN 60204-1:1997	Safety of machinery – Electrical equipment
Human Expos.		EN 50364 : 2001	Limitation of human exposure to electromagnetic fields
		EN 50371 : 2002	Limitation of human exposure to electromagnetic fields (10MHz-300GHz) – Generic Standard

Remarks: Only "x"-marked standards are relevant for the product and for this declaration of conformity! Concerning safety aspects, the general and the product specific warning and safety instruction in the product accompanying documents must also be regarded!

This declaration is based upon the respective technical documentation held by the manufacturer.

CE

Schruns, 30th March 2004

Gantner Instruments
Test and Measurement GmbH
Montafonerstr. 8, A-6780 Schruns
www.gantner-instruments.com

Werner Ganahl, General Manager

Gantner Instruments Test & Measurement GmbH
Montafonerstr. 8 A-6780 Schruns
Tel. +43 5556 - 73748 - 410
Fax +43 5556 - 73748 - 419
Firmenbuch FN 245178t UID ATU57139723

Industriestr. 12 D-64297 Darmstadt
Tel. +49 6151 - 95136 - 0
Fax +49 6151 - 95136 - 29
HRB 5989 AG Darmstadt USt-IDNr. DE 081401543

Geschäftsführung: Werner Ganahl, Reinhard Kolnar
office@gantner-instruments.com
www.gantner-instruments.com

e.bloxx A1-1, A1-4, A1-8
DECLERATION OF CONFORMITY

Gantner instruments

Gantner instruments

Konformitätserklärung – Declaration of Conformity – Déclaration de Conformité

The undersigned, representing:

Gantner Instruments Test & Measurement GmbH
Montafonerstr. 8 – A-6780 Schruns /Austria
tel: +43/5556-73748-410 – www.gantner-instruments.com

herewith declares, that the product:

e.bloxx A1-8
Certificate Ref No: 040330WG-03

is in conformity with the following EC directive(s), including all applicable amendments:

	Directives	Short Title
X	89 / 336 / EEC	EMC Directive
	99 / 5 / EEC	R&TTE Directive
	73 / 23 / EEC	Low Voltage Directive
	98 / 37 / EEC	Machinery Directive
	99 / 519 / EEC	Limitation of human exposure to electromagnetic Fields

Only "x"-marked directives are relevant for the product and for this declaration of conformity!

and that the standards and/or technical specifications referenced below have been applied:

		Standards	Short Title
EMC		EN 61000-6-1 : 2001	Generic immunity standard for residential, commercial and light-industrial environments
	X	EN 61000-6-2 : 1999	Generic immunity standard for industrial environments
		EN 61000-6-3 : 2001	Generic emission standard for residential, commercial and light-industrial environments
	X	EN 61000-6-4 : 2001	Generic emission standard for industrial environments
	X	EN 61326: 1997+A1+A2	Electrical equipment for measurement, control and laboratory use – EMC requirements
R&TTE		EN 300220-1/3 : 2000	Electromagnetic compatibility for Short Range Devices (SRDs) from 25 to 1000 MHz
		EN 300330-1/2 : 2001	Electromagnetic compatibility for Short Range Devices (SRDs) from 9 kHz to 25 MHz
		EN 301489-1/3 : 2001	Electromagnetic compatibility for Short Range Devices (SRDs) from 9 kHz to 40 GHz
Safety		EN 61010 : 2001	Safety requirements for electrical equipment for measurement, control and laboratory use
		EN 60950 : 2000	Safety requirements for information technology equipment
		EN 60335 : 2002	Safety of household and similar electrical appliances
		EN 60601 : 1988	Safety requirements for medical electrical equipment
Machinery		EN 292-1/2: 1991	Safety of machinery – Basic concepts, general principles for design
		EN 954-1: 1996	Safety of machinery – Safety-related parts of control system
		EN 60204-1:1997	Safety of machinery – Electrical equipment
Human Expos.		EN 50364 : 2001	Limitation of human exposure to electromagnetic fields
		EN 50371 : 2002	Limitation of human exposure to electromagnetic fields (10MHz-300GHz) – Generic Standard

Remarks: Only "x"-marked standards are relevant for the product and for this declaration of conformity! Concerning safety aspects, the general and the product specific warning and safety instruction in the product accompanying documents must also be regarded!

This declaration is based upon the respective technical documentation held by the manufacturer.

CE

Schruns, 30th March 2004

Gantner Instruments
Test and Measurement GmbH
Montafonerstr. 8, A-6780 Schruns

Werner Ganahl, General Manager

Gantner Instruments Test & Measurement GmbH
Montafonerstr. 8 A-6780 Schruns
Tel. + 43 5556 - 73784 - 410
Fax + 43 5556 - 73784 - 415
Firmenbuch FN 248174a UID ATU57729227

Industriestr. 12 D-64297 Darmstadt
Tel. + 49 6151 - 95136 - 0
Fax + 49 6151 - 95136 - 26
HRB 6189 AG Darmstadt UID DE216015743

Geschäftsführung: Werner Ganahl, Reinhard Kohler
office@gantner-instruments.com
www.gantner-instruments.com

e.blocx A1-1, A1-4, A1-8

www.gantner-instruments.com

Gantner Instruments Test & Measurement GmbH
Montafonerstraße 8 • A-6780 Schruns/Austria
Tel.: +43 (0)5556-73784-410 • Fax: +43 (0)5556-73784-419
E-Mail: office@gantner-instruments.com

3. Módulo entradas/salidas E.Bloxx D2-1

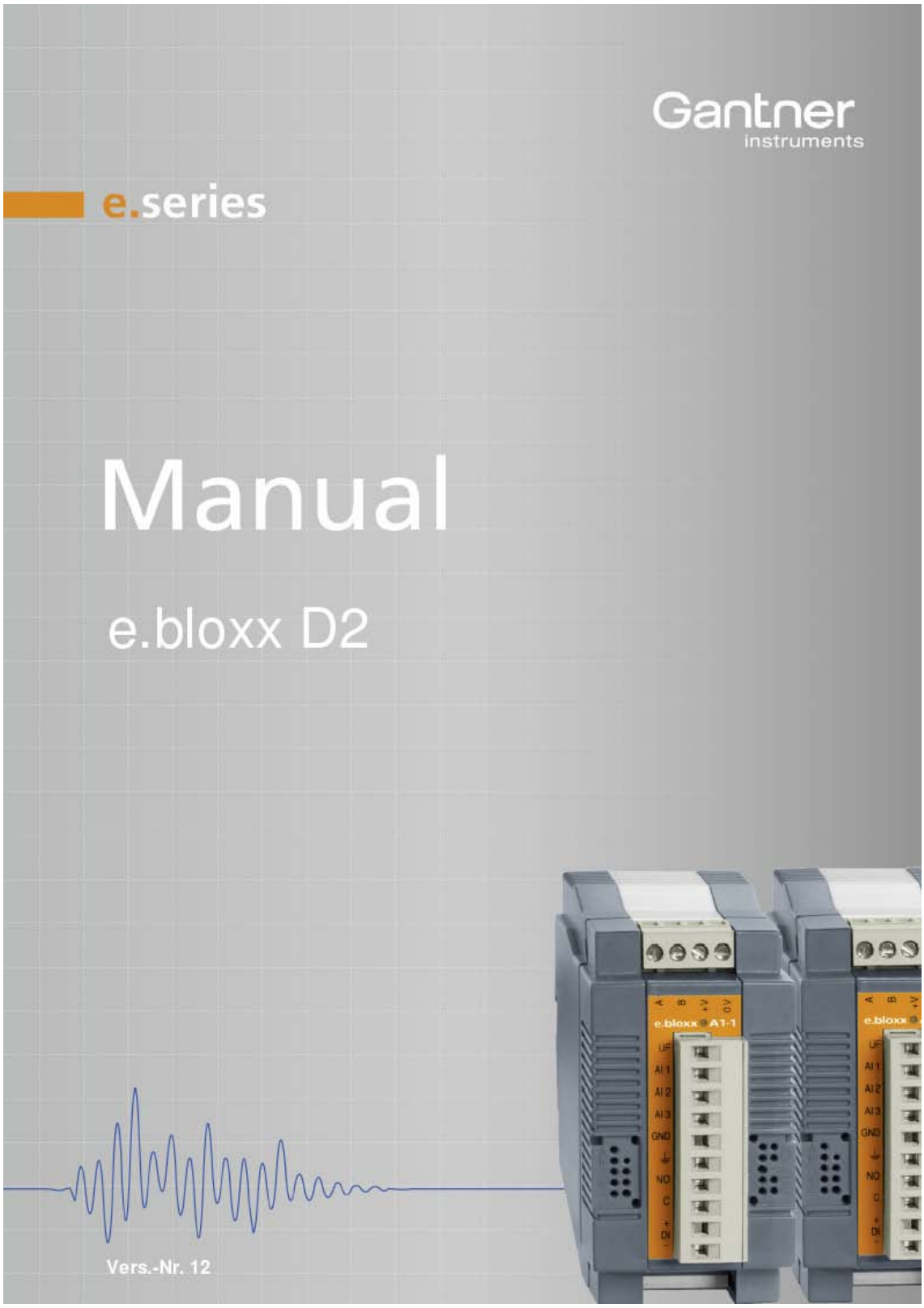


TABLE OF CONTENTS

1. ABOUT THIS MANUAL 6

2. MODULE DESCRIPTION 7
2.1. System Overview 7
2.2. Types of Modules 8
2.3. Module Parts 9
2.4. Front-LED 10
2.5. DC-Isolation 10
2.6. Functional Diagram 10

3. MOUNTING E.BLOXX AND CONNECTING WIRES 11
3.1. Environmental Conditions 11
3.2. Connection Technique 11
3.3. Power Supply 11
3.4. Bus Connection 12
3.5. Shielding 17
3.6. Rapid Bus Link Plug 17

4. MEASUREMENTS 19
4.1. General 19
4.2. Digital Output - Host Controlled 19
4.3. Digital Output - Process Controlled 20
4.4. Digital Output - Output Set 21

5. CONFIGURATION 22
5.1. General Information about Configuration Software ICP 100 22
5.2. Setting Address and Baud Rate of an e.bloxx D2-1 22
5.3. Settings for an e.bloxx D2-1 23
5.4. Definition of Variables 24

6. SPECIFICATIONS 25
6.1. Power Supply 25
6.2. Digital Outputs 25
6.3. Communication Interface 25
6.4. Mechanical 25
6.5. Connection 26
6.6. Environmental Conditions 26

7. DECLERATION OF CONFORMITY 27

1. ABOUT THIS MANUAL

This manual describes the installation and setup of the e.bloxx D2-1 module.

The following information can be found in this manual:
- Description of the e.bloxx system with detailed information on the hardware and module features.
- Installation description of the module and how it is connected to the power supply and bus lines.
- Description of the different types of measurement or outputs respectively.
- A short introduction on how the e.bloxx D2-1 module is configured with the CONFIGURATION SOFTWARE ICP 100. This software has an integrated help including a detailed description of the configuration process.
- Possible errors and its solutions.
- Technical specifications of the module.

2. MODULE DESCRIPTION

System Overview

The e.bloxx modules have been developed for the industrial and experimental testing technology, especially for the multi-variable measurement of electrical signals of thermal or mechanical data at test beds and test sites.

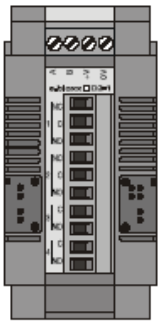


Figure 2.1 - e.bloxx D2-1 Front View

The e.bloxx D2-1, which is described in this manual, is an 8-variable (channel) module for output of digital status signals via relays. It is part of a whole product line of different e.bloxx which differ by their number and type of inputs and outputs.

Due to the fast and precise signal conditioning the e.bloxx modules produce reliable and exact measurement data.

Standardised interfaces guarantee the integration of up to 127 modules into a single network.

With the e.gate module very high data rates via Profibus-DP and Ethernet can be realized. The customer-specific signal processing supplements the standard conditioning of the single e.bloxx modules.

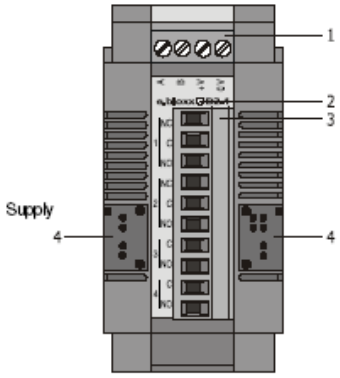
Types of Modules

There are several types of e.bloxx that differ in their number and type of analog and digital inputs and outputs.

	A1-1	A1-4	A1-8	A3-1	A3-4	A4-1	A4-4	A5-1	A5-1 CR	A6-1	A6-3	A6-2	A9-1	D1-1	D1-4	D2-1
Voltage Supply	10 - 30 VDC															
Power Consumption [W]	1,5	6	12	1,5	6	1,5	6	1,5	1,5	2	6	5	2,3	1,5	6	2
Variable / Channels	8	32	64	8	32	8	32	8	8	8	24	16	8	8	32	8
Analog Inputs	1	4	8	4	16	4	16	2/3/6	2	1	3	1	-	-	-	-
Analog Outputs	-	-	-	-	-	-	-	-	-	1	3	2	4	-	-	-
Relay Outputs	1	4	8	-	-	-	-	-	-	-	-	-	-	-	-	4
Digital Inputs	1	4	8	1	4	-	-	1	1	1	3	6	1	8	32	-
Digital Outputs	-	-	-	1	4	-	-	1	1	1	3	4	1	8	32	-
Fieldbus Interface	RS 485															
Protocols	ASCII - Modbus-RTU - Profibus-DP - LocalBus															
Quantity to measure / Sensor Principle																
Voltage	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	-
Current	x	x	x	-	-	-	-	-	-	-	-	-	-	-	-	-
Resistance	x	x	x	-	-	-	-	-	-	-	-	-	-	-	-	-
Pt100 / Pt1000	x	x	x	-	-	-	-	x	-	-	-	-	-	-	-	-
Cryo Sensor	-	-	-	-	-	-	-	-	x	-	-	-	-	-	-	-
Thermocouple	x	x	x	-	-	x	x		-	-	-	-	-	-	-	-
Strain Gauge Full Bridge	x	x	x	-	-	-	-	-	-	x	x	x	-	-	-	-
Strain Gauge Half Bridge	-	-	-	-	-	-	-	-	-	x	x	x	-	-	-	-
Strain Gauge Quarter Br.	-	-	-	-	-	-	-	-	-	x	x	x	-	-	-	-
Inductive Full Brigde	-	-	-	-	-	-	-	-	-	x	x	x	-	-	-	-
Inductive Half Bridge	-	-	-	-	-	-	-	-	-	x	x	x	-	-	-	-
LVDT	-	-	-	-	-	-	-	-	-	x	x	x	-	-	-	-
Potentiom. Transducer	x	x	x	-	-	-	-	-	-	-	-	x	-	-	-	-
Piezoresist. Transducer	-	-	-	-	-	-	-	-	-	-	-	x	-	-	-	-
Status	x	x	x	x	x	-	-	x	x	x	x	x	x	x	x	-
Frequency	-	-	-	-	-	-	-	-	-	-	-	x	-	x	x	-
Counter	-	-	-	-	-	-	-	-	-	-	-	x	-	x	x	-

Table 2.1. - Characteristics of the e.bloxx Modules

Module Parts



1 ... Pluggable Screw-Type Terminal Strip for Connection of RS 485 Bus and Power

2 ... Power/Error-LED (red/green)
3 ... Pluggable Screw-Type Terminal Strip for Sensor Connection
4 ... Rapid Bus Link Plugs

Figure 2.2. - Parts of the e-bloxx D2-1

Terminal Strip for RS 485 and Power Supply

Terminal	Description
A	RS 485 Bus Interface A
B	RS 485 Bus Interface B
+V	Power Supply +
0V	Power Supply -

Table 2.2. - Description of Terminal Strip for RS 485 Bus and Power Supply

Terminal Strip for Sensor Connection

Terminal	Description
1 | NC	Relay Output 1 – Normal Closed Contact
1 | C	Relay Output 1 – Common Contact
1 | NO	Relay Output 1 – Normal Open Contact
2 | NC	Relay Output 2 – Normal Closed Contact
2 | C	Relay Output 2 – Common Contact
2 | NO	Relay Output 2 – Normal Open Contact
3 | C	Relay Output 3 – Common Contact
3 | NO	Relay Output 3 – Normal Open Contact
4 | C	Relay Output 4 – Common Contact
4 | NO	Relay Output 4 – Normal Open Contact

Table 2.3. - Description of Terminal Strip for Sensor Connection

2.4. Front-LED

The LED at the front of the e.bloxx modules provides the following information:

LED green	Module works well, no signal overflow, no communication error…
LED red	general error like signal overflow, broken sense leads
LED red + short off period	general error like signal overflow, broken sense leads + communication timeout
LED green + short red flash	Signal ok + communication timeout
LED red fast flashing	global error, no suitable firmware

Notice: The LED will get red when the signal leaves the selected range and the error checking is activated (see ICP 100 column Range/Error).

2.5. DC-Isolation

The power supply and bus interface section are DC-isolated from each other.

2.6. Functional Diagram

The e.bloxx D2-1 can be described by the following figure.

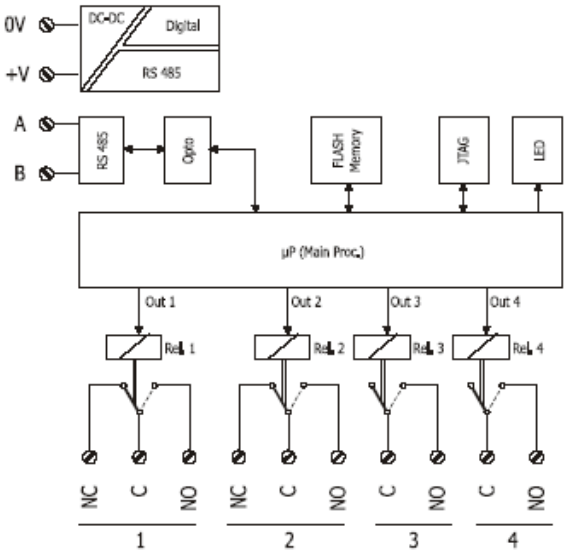


Figure 2.3. - Functional Diagram of the e.bloxx D2-1

3. MOUNTING e.bloxx AND CONNECTING WIRES

Environmental Conditions

The e.bloxx modules are protected against water and dirt according IP 20. If required by the conditions of the operating site the module has to be installed accordingly, e.g. in a water-resistant or waterproof case, compliant with the regulations of electrical engineering.
For the allowed ambient temperatures for the e.bloxx D2-1 see the Technical Specifications at the end of this manual.

Connection Technique

The wires are connected to the e.bloxx D2-1 via screw-type terminals. The captive terminal screws are part of the terminal strips. All terminal strips are of plug-in type and can be detached from the module.

Not more than 2 leads should be connected with one clamp. In this case both leads should have the same conductor cross-section. For the precise clamping of stranded wire we recommend the use of wire-end ferrules.

Notice: Connecting wires respectively the plugging-in and –out of the terminal strip is only allowed with modules in power-off status.
In order to prevent interference with sensors, signals and modules, shielded cables have to be used for the power supply, bus connection and signal lines.

We strongly recommend using a single screened cable each input signal. To use more signals in one cable could generate interacting influences.

Notice: For optimal performance the e.bloxx modules must be grounded properly. This is achieved by utilizing the Ground/Earth screw on the back of each e.bloxx module. The screen of the sensor cable has to be grounded at the same potential.

Power Supply

Non-regulated DC voltage between +10 and +30 VDC is sufficient for the power supply of the e.bloxx D2-1. The input is protected against excess voltage, current and polarity connecting error. The power consumption remains approximately constant over the total voltage range, due to the integrated switching regulator.

Due to their low current consumption the modules can also be remotely supplied via longer lines. Several modules can be supplied in parallel within the permissible voltage range and drop in the lines. If required, the supply lines together with the bus line may be incorporated in one cable.

In order not to overload the modules power supply needlessly and to avoid unnecessary line troubles, a separate power supply is recommended for sensors with a large current drain.

The distribution voltage for the e.bloxx modules has to be protected by a fuse with maximum 1 A (inert). The modules have an internal fuse (reversible) for protection against excess voltage, excess current and wrong polarity.

Notice: It depends on the power supply unit and its noise and internal grounding issues whether it is helpful to connect earth of the power supply unit with ground/earth of the e.bloxx module.

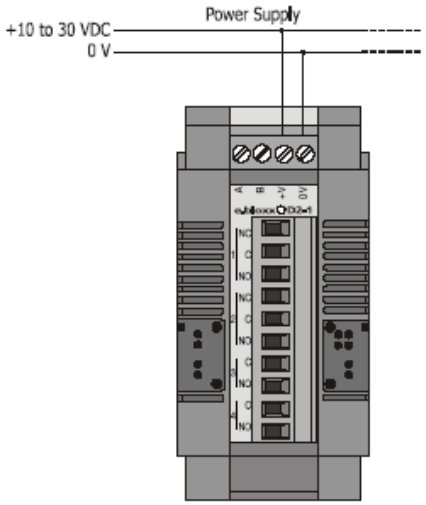


Figure 3.1. - Power Supply of the e.bloxx Modules

Bus Connection

Only the connection of the e.bloxx modules to the bus is described in here. A detailed description of the bus and the communication of the e.bloxx modules can be found in the Communication Guide of the e.bloxx modules.

The e.bloxx modules have an RS 485 bus interface for connection to the serial fieldbus. The bus has to be terminated on both sides with a characteristic impedance. The maximum line length depends on the transmission speed (refer to the Communcation Guide for details) and can never be higher than 1.2 km per bus segment or 4.8 km via a physical bus string by using 3 repeaters. A maximum of 32 devices are possible at each bus segment and up to 127 devices via a physical bus string.

Wiring

In general, the e.bloxx is connected to the bus by connecting both signal leads A and B of the incoming bus cable and A' and B' of the outgoing bus cable together to one terminal on the module (Figure 3.2). Alternatively, the bus can also be connected by a "stub cable" (Figure 3.3). This guarantees that the bus connection to other modules remains in place, even if one module has to be exchanged, due to the removable terminal strip.

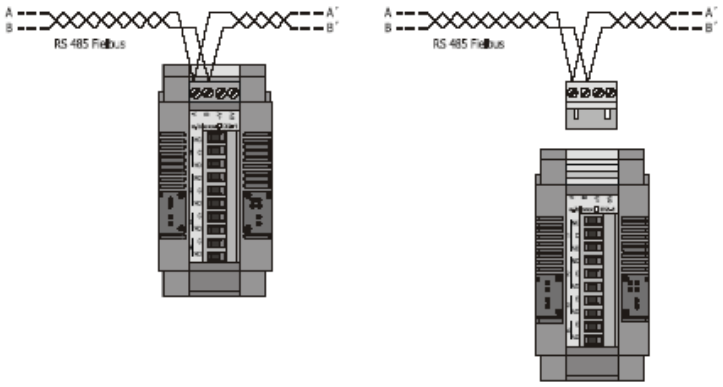


Figure 3.2 - Bus Connection of an e.bloxx D2-1 to the RS 485 Fieldbus with Derivation

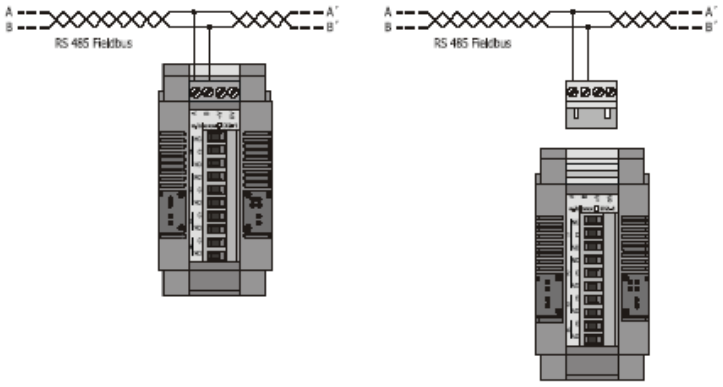


Figure 3.3. - Bus Connection of an e.bloxx D2-1 to the RS 485 Fieldbus via Stub Cable

The stub-cable should be as short as possible, not longer than 30 cm.

Notice: The terminal designations A and B of all modules of the e.bloxx series are exchanged compared with the PROFIBUS-definitions. Consequently, in multi-vendor systems the bus lines A and B have to be exchanged when connecting them to the e.bloxx.

Bus Structure

The bus structure is a line structure where each bus segment will be terminated with characteristic impedance on both ends. Branches can be set up by means of a bi-directional signal amplifier, so-called repeaters. Other types of branches are not permitted (no tree topology). The max. stub-length to a user must not exceed 30 cm.

The following figures show a few examples of possible bus topology structures. The meanings of the symbols are as follows:

... Bus User
... Repeater
... Bus Termination

Figure 3.4. - Simple Line Structure

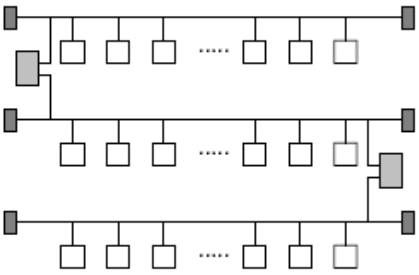


Figure 3.5. - Extended Line Structure

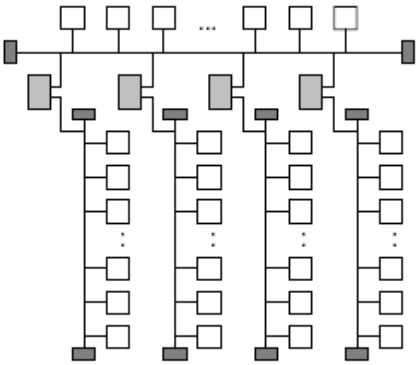


Figure 3.6. - Line Structure with Branches

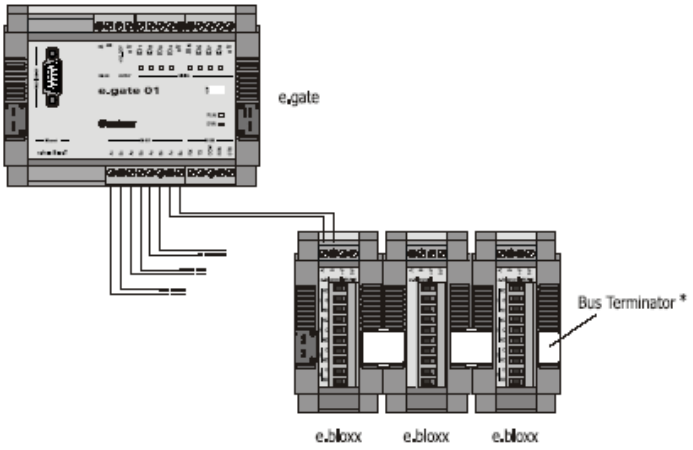


Figure 3.7. - e.bloxx D2-1 connected to e.gate

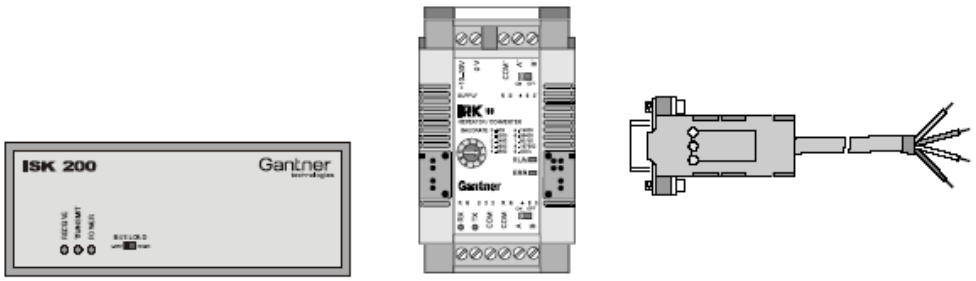
* ... If e.bloxx modules are used together with an e.gate, which is used to collect the data of all connected e.bloxx modules and processes them for fast transmission via the further network, a bus termination must be connected to the last e.bloxx in each bus line.

Bus Connection to PC

The bus interface of the e.bloxx is based on the RS 485 standard. Since most hosts are "only" equipped with RS 232 interfaces, an interface converter or a plug-in board with RS 485 drivers is required for conversion purposes.

Gantner Instruments Test & Measurement GmbH offers a compact interface converter, called ISK 200, with an integrated power supply and automatic baud rate detection. The power supply, bus connection and a separate 24 VDC-output are DC-isolated. Therefore, the interface converter ISK 200 is also applicable as a power supply for remote applications. Additionally, the interface converter ISK 200 features the option of connecting the required bus termination via a switch. The converter is designed to be used as a desk device.

Another module IRK 100 from Gantner Instruments Test & Measurement GmbH is available which may be used as an RS 485 repeater or RS 485/RS 232 converter. The baud rate can be adjusted at the IRK 100. Also, for this module the required bus termination may be connected with a switch. The Repeater/ Converter IRK 100 has a snap-on mounting mechanism for the installation on standard profile rails (DIN rail) 35 mm according to DIN EN 50022.



Interface Converter ISK 200 | Repeater/Converter IRK 100 | Interface Converter ISK 101

Figure 3.8. - Interface Converters ISK 200, IRK 100 and ISK 101

Bus Connection to Profibus-DP

For the installation of the bus cable and bus interface, 9-channel D-subminiature plugs and sockets are used. The pin assignment for the RS 485 connection according to PROFIBUS is given in Table 3.1.

Plug	Pin	RS 485 Notation	Signal	Identification
 DB 9	1	-	Shield	Shield, Protective Ground
	2	-	RP	Reserved for Power
	3	B / B´	RxD/TxD-P	Receive/Transmit-Data-P
	4	-	CNTR-P	Control-P
	5	C / C´	DGND	Data Ground
	6	-	VP	Voltage Plus
	7	-	RP	Reserved for Power
	8	A / A´	RxD/TxD-N	Receive/Transmit-Data-N
	9	-	CNTR-N	Control-N

Table 3.1. - Pin Assignment D-Subminiature Plug According to PROFIBUS

The signal leads A and B (and Shield) are mandatory for a (shielded) connection. Additional signal leads may be installed if required.

Notice: Due to the fact that the RS 485 interface is used for different protocols, in case of using Profibus-DP the leads A and B has to be crossed.

Bus Termination at the e.bloxx Modules

In order to avoid signal reflections on the bus, each bus segment has to be terminated at its physical beginning and at its end with the characteristic impedance. A terminating resistor is installed between the bus leads A and B for this purpose. In addition, the bus lead A is connected via a pull-up resistor to potential (VP) and the bus lead B is connected via a pull-down resistor to ground (DataGround). These resistors provide a defined quiescent potential in case there is no data transmission on the bus. This quiescent potential is level high.

The e.bloxx modules have these bus termination resistors built in. They can be connect to the bus by plugging the Bus *Termination Plug IBT 100*, which is available as accessory, into the rapid bus link plug on the front side of the module. Instead of the bus termination plug *IBT 100*, also separate jumpers may be used for the bus termination. In this case, it is mandatory that the jumper clips are installed as indicated below, and that the bus leads or the bus termination are not short-circuited by mistake.

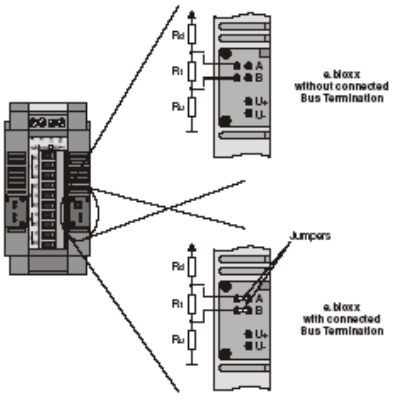


Figure 3.9. - Bus Termination at the e.bloxx Modules

Shielding

In case of increased interference, such as in industrial areas, we recommend shielding of bus and signal cables. In general, the shield should be connected to the protective earthing (not Data Ground!) at each bus connection. If necessary, the shield should also be applied along the course of the cable several times. For shorter distances, e.g. with stub cables, the interference response is often improved if the shielding is only applied to the stub cable exit.

Bus users such as controllers (PLCs), computers (PCs), repeaters and interface converters (ISK), etc., generally feature the possibility of applying the shield directly to the appliance or to separate shield rails. Shield rails offer the advantage of preventing possible interfering signals from reaching the appliance. The shields, which are connected to protective earthing conduct interference signals off before reaching the module.

The e.bloxx do not have a direct shield connection at the module. Here the shield of the bus cable can be connected to earth e.g. by so-called shield clamps.

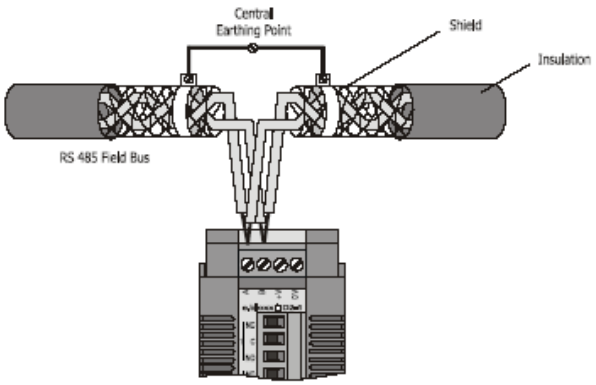


Figure 3.10. - Earthing of the Bus Line Shield at an e.bloxx

Notice: The shielding screen must not be connected to the ground (0V) of the power supply and it should always be connected to earth with a large surface and low-inductance.

Rapid Bus Link Plug

The e.bloxx D2-1 has a plug on the left and on the right side which allow to connect the bus and power supply from one module to the next with a Rapid Bus Link Plug (type designation: *ICM 100*). This kind of connecting bus and power supply is particularly advantageous if several modules are mounted on one common profile rail side by side. In this case, only the terminal of one module has to be connected. Furthermore, various modules of the e.bloxx series may be connected with the Rapid Bus Link Plug.

Notice: The current flowing through the Rapid Bus Link Plug Jack and the e.bloxx must not exceed 1 A. Thus, the power supply should preferably be connected to the middle of several modules and no more than 5 pieces of e.bloxx may be connected via the Rapid Bus Link Plug *ICM 100* in one line.

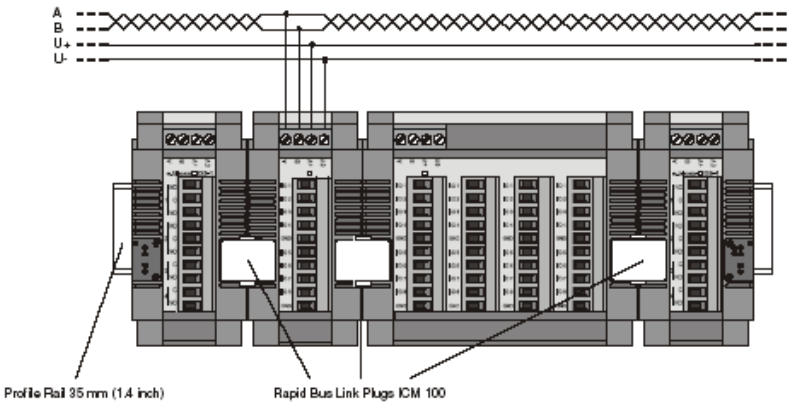


Figure 3.11. - Connection of four e.bloxx Modules with Rapid Bus Link Plugs ICM 100

4. MEASUREMENTS

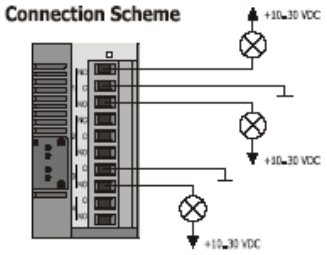
General

The e.bloxx D2-1 has 4 digital outputs. The output 1 and 2 are relays with 3 connections (NO, C and NC) and the outputs 3 and 4 are relays with just one switch or two connections respectively (NO, C). The configuration of the outputs is done with the Configuration Software ICP 100 as required by the application.

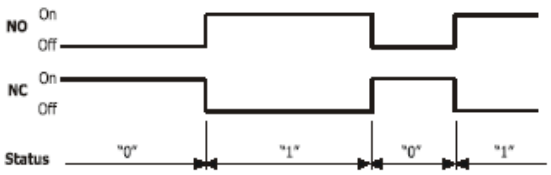
Digital Outputs
The Digital Output Variable supports:
- digital status output, host controlled
- digital status output, process controlled
- digital status output set

It is possible to control the relays of the digital outputs via bus (host controlled) or autonomously by the e.bloxx D2-1 itself. Therefore the e.bloxx D2-1 will check one or more of its variables due to certain thresholds (process controlled). A typical case of application would be e.g. the local output of an acoustic or optical signal in case a certain event happened. Furthermore it is possible to set all 4 digital outputs in common (status set).

Digital Output - Host Controlled



Picture 4.1 - Digital Output as State Output



Picture 4.2 - Signal Diagram

With a digital output set to "State" the digital output is set according to the status information received via bus.

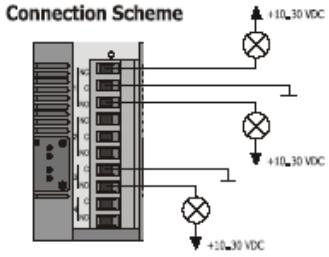
At the outputs 1 and 2 two contacts of the internal relays can be connected. At outputs 3 and 4 only one contact is connectable. "C" is the centre contact. The contact "NO" is open if the output is not active ("0") whereas the contact

"NC" is closed if the output is not active. Both contacts will change from open to closed or reverse respectively if the output is set active ("1").

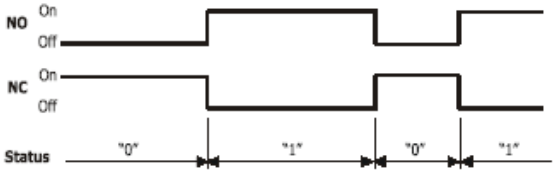
The voltage connected at the relay outputs can range from 10 to 30 VDC / 3 A or 250 VAC / 3 A (see also specifications). It has to be either supplied externally or be picked up by the power supply of the e.bloxx D2-1 (DC).

The status of the digital output can be scanned as 1/0 information via bus.

Digital Output - Process Controlled



Picture 4.3 - Digital Output as Process Output



Picture 4.4 - Signal Diagram

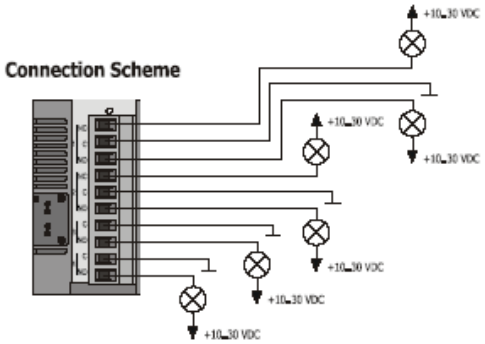
With a digital output set to "Process Out" the digital output monitors other sensor variables as to constraints (threshold values). The digital output is set if one or several threshold conditions are fulfilled. The user can freely define the constraints. The user can also preset the logical signal level (see also the *Configuration Software ICP 100*).

At the outputs 1 and 2 two contacts of the internal relays can be connected. At outputs 3 and 4 only one contact is connectable. "C" is the centre contact. The contact "NO" is open if the output is not active ("0") whereas the contact "NC" is closed if the output is not active. Both contacts will change from open to closed or reverse respectively if the output is set active ("1").

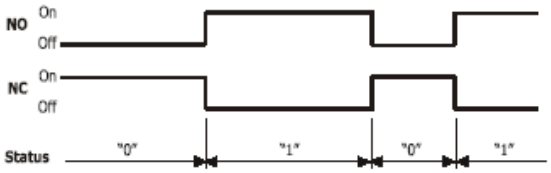
The voltage connected at the relay outputs can range from 10 to 30 VDC / 3 A or 250 VAC / 3 A (see also specifications). It has to be either supplied externally or be picked up by the power supply of the e.bloxx D2-1 (DC).

The status of the digital output can be scanned as 1/0 information via bus.

Digital Output - Output Set



Picture 4.5 - Digital Output as Output Set



Picture 4.6 - Signal Diagram

With a digital output set to "Output Set" all 4 digital outputs are set at once according to the status information received via bus.

At the outputs 1 and 2 two contacts of the internal relays are connected. At outputs 3 and 4 only one contact is connectable. "C" is the centre contact. The contact "NO" is open if the output is not active ("0") whereas the contact "NC" is closed if the output is not active. Both contacts will change from open to closed or reverse respectively if the output is set active ("1").

The voltage connected at the relay outputs can range from 10 to 30 VDC / 3 A or 250 VAC / 3 A (see also specifications). It has to be either supplied externally or be picked up by the power supply of the e.bloxx D2-1 (DC).

The status of the digital output can be scanned as 1/0 information via bus.

5. CONFIGURATION

General Information about Configuration Software ICP 100

The e.bloxx modules can be configured with the Configuration Software ICP 100. This software includes all functions to set the module parameters like baud rate, measurement rate, etc. and to define the output functions like the type of output function, threshold values, etc..
The Configuration Software ICP 100 also includes a function to display the values of the variables in real-time. There are also several software from other companies, that are adapted to the specific measurement tasks.
In the Configuration Software ICP 100 the two register cards "Variable Settings" and "Module Settings" will be displayed if you are configuring an e.bloxx that is not online and also the two additional register cards "Info" and "Measure" when configuring an online e.bloxx.

- On the register card "Info" several module information will be displayed.
- On the register card "Measure" the variable values of the online e.bloxx D2-1 will be displayed in real time.
- On the register card "Variable Settings" the different variables of the e.bloxx D2-1 can be configured. This will be done in the Variable Settings Table being displayed on this register card.
- On the register card "Module Settings" different general settings like the baud rate, address, etc. can be defined for each e.bloxx.

This manual only gives a brief description on how to set up and configure an e.bloxx module. A detailed description of all the functions of the Configuration Software ICP 100 is included in the help function of the software.

Setting Address and Baud Rate of an e.bloxx D2-1

Before a control (PLC) or a computer (PC) can interchange data with an e.bloxx via the bus, address and baud rate of the e.bloxx have to be defined. The following points have to be taken into consideration in this connection:
- All devices on the bus have to be adjusted to the same baud rate.
- The same address must not appear twice in the bus topology.

The setting variants for the bus parameters for e.bloxx are:

Bus Parameter	ASCII Protocol	MODBUS Protocol	Profibus-DP Protocol	LOCAL-BUS Protocol
Address	1 127	1 127	1.....126	1 127
Baud Rate	19,200 bps	19,200 bps	19,200 bps	19,200 bps
	38,400 bps	38,400 bps	-	38,400 bps
	57,600 bps	57,600 bps	-	57,500 bps
	93,750 bps	93,750 bps	93,750 bps	93,750 bps
	115,200 bps	115,200 bps	-	115,200 bps
	-	-	187,500 bps	187,500 bps
	-	-	500,000 bps	500 kbps
	-	-	1.500 kbps	1.500 kbps

Table 5.1 - Setting variants for address and baud rate for the e.bloxx

If no other specifications are made on delivery, the e.bloxx have address 1 and baud rate 1.5 Mbps as default. The adjustment can be changed via the bus by means of the *Configuration Software ICP 100.*

Adjustment via bus by means of the Configuration Software ICP 100:

The address and baud rate of an e.bloxx D2-1 can be set in the Configuration Software ICP 100. On the dialog box "Module Information" the address and baud rate of the actual e.bloxx D2-1 is displayed. After changing these settings, the new settings have to be loaded into the e.bloxx D2-1 in order to take effect. To do this the menu item **Send to Module** or **Send to Module as...** in the menu **File** or the corresponding button ( ) in the icon bar has to be selected.

Notice: The address 0 is provided for the PC in case of a transmission via PROFIBUS-DP. This address can therefore not be assigned to an e.bloxx D2-1. Also the address 127 is reserved for broadcast transmission in the PROFIBUS-DP protocol and may only be assigned for these cases.

Settings for an e.bloxx D2-1

On the register card "Module Settings" the following settings of an e.bloxx D2-1 can be defined.

- Location: Description of each e.bloxx D2-1.
- User Name: Possibility to enter the name of the person that has configured the e.bloxx D2-1.
- Config. Date: Displays the date of configuration.
- Address: Address of the online e.bloxx D2-1. Will only be displayed if e.bloxx D2-1 is online.
- Protocol: Bus protocol that is used for communication between PC and e.bloxx D2-1. Will only be displayed if e.bloxx D2-1 is online. In the configuration software ICP 100 only the LocalBus protocol is displayed. Nevertheless all the protocols mentioned in chapter 2.2 are available and the e.bloxx D2-1 uses the required protocol automatically.
- Character Format: Determines the number of data, parity and stop bits for transmission between PC and e.bloxx D2-1. Will only be displayed if e.bloxx D2-1 is online. With the e.bloxx D2-1 the character format is fixed to 8E1.
- Answer Delay: Determines how long an e.bloxx D2-1 will wait before it sends an answer to a host request.
- Timeout: A timeout means that there is no communication with the module during the time period that is set here. All host-controlled functions (Digital Status and Set Output variables and Setpoint variables) can be defined to pass into a safe, definable status. As soon as the communication recommences, the values are assumed again, depending on the configuration.
- Special Data: If a special program (firmware) is loaded in the e.bloxx D2-1 it may need some special data that can be input here.

After changing some of these settings, the new settings have to be loaded into the corresponding e.bloxx D2-1, so they can take effect. Therefore select the menu item **Send to Module** or **Send to Module as...** in the menu **File** or the corresponding button ( ) in the icon bar.

Definition of Variables

Up to 8 variables (real + virtual) can be defined for an e.bloxx D2-1. They define how the signals at the in- and outputs of the e.bloxx D2-1 will be processed. The value of every variable can be read out via the fieldbus. The variables are defined in the Variable Settings Table in the Configuration Software ICP 100.

The Variable Settings Table is displayed on the corresponding register card in the Configuration Software ICP 100.

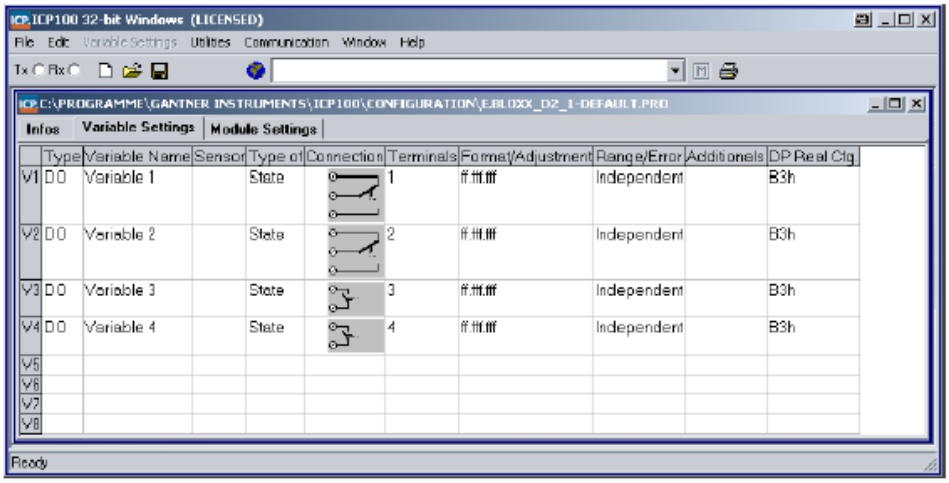


Figure 5.1. - Example for Variable Settings Table

Here all 8 possible variables will be listed. To define a new variable just click on a free line in the table or to change the type of variable click on the first column *Type* in the corresponding line. In both cases a dialog box will be opened where the type of the new variable has to be selected. There are 4 different types of variables:

- Digital Output Variable: Used to output digital status signals via the relay outputs of the e.bloxx D2-1. In the column *Type of Measurement* it is possible to select the measurement function (state (= host controlled), process Out, Output Set).
- Arithmetic Variable: With this variable it is possible to perform calculations with the actual values of other variables and with constant values. The results of the calculations are assigned to the arithmetic variable and so arithmetic variables can also be used by other arithmetic variables for calculations.
- Alarm Variable: An Alarm Variable can be used to monitor another variable and to generate an alarm message if one of up to 4 definable thresholds are exceeded. The alarm messages can be read via bus.
- Setpoint Variable: The value of this variable can be set via bus. This way it is possible to set a value via bus which can be used by an arithmetic variable for further processing, e.g. to set a factor for measurement by the user.

The settings for all defined variables will be displayed in the corresponding column of the variables. To change these settings click on the corresponding field in the Variable Settings Table.

6. SPECIFICATIONS

All following data are valid after a warm-up time of approx. 45 minutes.

Power Supply

Power Supply:	10 VDC up to 30 VDC Overvoltage and overload protection
Power Consumption:	Approx. 2.5 W

Digital Outputs

Arrangement:	2 x 1a, 2 x 1a1b
Initial Contact Resistance:	Max. 30 mΩ (by voltage drop 6 VDC / 1 A)
Contact Material:	Gold flash over silver alloy
Nominal Switching Capacity:	250 VAC / 3 A 30 VDC / 3 A (resistive)
Switching Power:	Max. 150 W, 1250 VA
Switching Voltage:	Max. 250 VAC, 30 VDC
Switching Current:	Max. 3 A
Expected Life Mechanical:	5×10^7 (min. operations)
Expected Life Electrical (Resist.):	10^5 (250 VAC / 3 A resp. 30 VDC / 3 A)

Communication Interface

Standard:	RS 485, 2-wire
Data Format:	8E1
Protocols:	ASCII, MODBUS, PROFIBUS-DP, LOCAL-BUS
Baud Rates:	
ASCII:	19.2, 38.4, 57.6, 93.75, 115.2 kbit/s
MODBUS:	19.2, 38.4, 57.6, 93.75, 115.2 kbit/s
PROFIBUS-DP:	19.2, 93.75, 187.5, 500, 1,500 kbit/s
LOCAL-BUS:	19.2, 38.4, 57.6, 93.75, 115.2, 187.5, 500, 1,500 kbit/s
Connectable Devices:	up to 32 without repeater up to 127 with repeater
Galvanic Isolation:	500 V

Mechanical

Case:	Aluminium and ABS
Dimensions (W x H x D):	45 x 90 x 83 mm (1.8 x 3.5 x 3.3 inch)
Protective System:	IP 20
Weight:	160 g
Mounting:	DIN EN-Rail

Connection

Plug-In Screw Terminals:	Wire cross-section up to 1.5 mm²
Rapid Bus Connector:	4-pin plug in ABS-housing

Environmental Conditions

Operating Temperature:	-20 °C up to + 60 °C (-4 °F to +140 °F)
Storage Temperature:	-30 °C up to +85 °C (-22 °F to +185 °F)
Humidity	0% up to 95% at +50 °C (+122 °F), non-condensing

7. DECLERATION OF CONFORMITY

Gantner instruments

Konformitätserklärung – Declaration of Conformity – Déclaration de Conformité

The undersigned, representing:

Gantner Instruments Test & Measurement GmbH
Montafonerstr. 8 – A-6780 Schruns /Austria
tel: +43/5556-73748-410 – www.gantner-instruments.com

herewith declares, that the product:

e.bloxx D2-1
Certificate Ref No: 040806WG-01

is in conformity with the following EC directive(s), including all applicable amendments:

Directives		Short Title
X	89 / 336 / EEC	EMC Directive
	99 / 5 / EEC	R&TTE Directive
X	73 / 23 / EEC	Low Voltage Directive
	98 / 37 / EEC	Machinery Directive
	99 / 519 / EEC	Limitation of human exposure to electromagnetic Fields

Only 'x'-marked directives are relevant for the product and for this declaration of conformity !

and that the standards and/or technical specifications referenced below have been applied:

Standards			Short Title
EMC		EN 61000-6-1 : 2001	Generic immunity standard for residential, commercial and light-industrial environments
	X	EN 61000-6-2 : 1999	Generic immunity standard for industrial environments
		EN 61000-6-3 : 2001	Generic emission standard for residential, commercial and light-industrial environments
	X	EN 61000-6-4 : 2001	Generic emission standard for industrial environments
	X	EN 61326: 1997+A1+A2	Electrical equipment for measurement, control and laboratory use – EMC requirements
R&TTE		EN 300220-1/3 : 2000	Electromagnetic compatibility for Short Range Devices (SRDs) from 25 to 1000 MHz
		EN 300330-1/2 : 2001	Electromagnetic compatibility for Short Range Devices (SRDs) from 9 kHz to 25 MHz
		EN 301489-1/3 : 2001	Electromagnetic compatibility for Short Range Devices (SRDs) from 9 kHz to 40 GHz
Safety	X	EN 61010 : 2001	Safety requirements for electrical equipment for measurement, control and laboratory use
		EN 60950 : 2000	Safety requirements for information technology equipment
		EN 60335 : 2002	Safety of household and similar electrical appliances
		EN 60601 : 1988	Safety requirements for medical electrical equipment
Machinery		EN 292-1/2: 1991	Safety of machinery – Basic concepts, general principles for design
		EN 954-1: 1996	Safety of machinery – Safety-related parts of control system
		EN 60204-1:1997	Safety of machinery – Electrical equipment
Human Expos.		EN 50364 : 2001	Limitation of human exposure to electromagnetic fields
		EN 50371 : 2002	Limitation of human exposure to electromagnetic fields (10MHz-300GHz) – Generic Standard

Remarks: Only 'x'-marked standards are relevant for the product and for this declaration of conformity! Concerning safety aspects, the general and the product specific warning and safety instruction in the product accompanying documents must also be regarded!

This declaration is based upon the respective technical documentation held by the manufacturer.

CE

Schruns, 8th August 2004

Gantner Instruments
Test and Measurement GmbH
Montafonerstr. 8, A-6780 Schruns
www.gantner-instruments.com

Werner Ganahl, General Manager

Gantner Instruments Test & Measurement GmbH
Montafonerstr. 8 – A-6780 Schruns
Tel. + 43 5556 - 73784 - 410
Fax + 43 5556 - 73784 - 419
Firmenbuch FN 245174a UID ATU57738723

Industriestr. 12 D-64297 Darmstadt
Tel. + 49 6151 - 95136 - 0
Fax + 49 6151 - 95136 - 25
HRB 81189 AG Darmstadt UID DE814015743

Geschäftsführung: Werner Ganahl, Reinhard Kehrer
office@gantner-instruments.com
www.gantner-instruments.com

e.bloxx D2-1

PartNo.: 268382

www.gantner-instruments.com

Gantner Instruments Test & Measurement GMBH
Montafonerstraße 8 • A-6780 Schruns/Austria
Tel.: +43 (0)5556-73784-410 • Fax: +43 (0)5556-73784-419
E-Mail: office@gantner-instruments.com

4. Servidor de vídeo y Webcam

WEB CAMERA
&
VIDEO SERVER

Quick Manual

(Version Eng_v1.3)

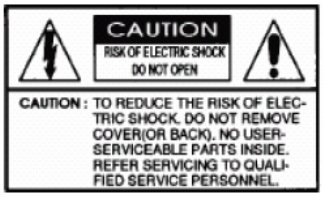
2002/05/02

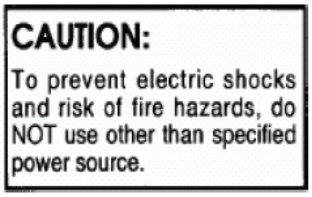
Warning

To prevent fire or shock hazard, do not expose the unit to rain or moisture.

The symbol is intended to alert the user to the presence of important operating and maintenance (servicing) instructions in the literature accompanying the appliance.

The symbol is intended to alert the user to the presence of uninsulated " dangerous voltage" within the product's enclosure that may be of sufficient magnitude to constitute a risk of electric shock to persons.





Warning - - The equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC Rules.

These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

Caution - - Any changes or modifications in construction of this device which are not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

Precautions

These pages list important information that will help to assure proper use of your primary and accessory equipment. Please read it carefully before operating your equipment and keep it in a handy place for future reference.

1. HEED WARNINGS – Adhere to all warnings on the appliance and in the operating instructions.

2. POWER SOURCES – Operate this equipment only from the type of power source indicated on the label. If you are not sure of the type of power being used, consult your dealer or local power company.

3. OVERLOADING – Do not overload wall units and extension cords, as this can result in a risk of fire or electric shock, Frayed power cords, damaged or cracked wire insulation, and broken plugs are dangerous. Periodically examine the cord and replace it if appearance indicates damage or deteriorated insulation.

4. POWER PLUG PROTECTION – Route the power supply cords so they cannot be walked on or pinched by items placed upon or against them.

5. VENTILATION – Do not block the slots and openings in the cabinet, or place the equipment on a bed, soft rug, or other similar surface.

6. ATTACHMENTS – Do not use attachments other than those specifically recommended by the equipment manufacturer as they may cause hazards.

7. TO PREVENT SHOCK HAZARD, DO NOT EXPOSE THIS UNIT TO RAIN OR MOISTURE – If you spill liquid on the unit, consult authorized service personnel. Moisture can damage internal parts. Do not use this equipment near sources of water.

8. ACCESSORIES – Use only with a manufacturer recommended card, stand, tripod, bracket, or table. The equipment may fall, causing serious injury to a child or adult, and serious damage to the appliance.

9. CLEANING THE OUTSIDE SURFACES - Unplug this equipment from the wall outlet before cleaners. Use a damp cloth for cleaning.

10. REQUIRING SERVICE - Unplug the equipment from the wall outlet and refer servicing to qualified service personnel.

11. SAFETY CHECK - Upon completion of any service or repairs to this equipment, ask the service technician to perform safety checks to determine that the equipment is in safe operating condition.

12. IMPORTANT NOTE TO THE INSTALLER - This installation should be made by a qualified service person and should conform to all local codes. In order to provide this product with protection against risk of unintentional operation by employees, customers, janitors and cleaners working on the premises, and from falling objects, building vibrations and similar causes, it is recommended this product be enclosed in an tamper-resistant lock box. Make sure that the lock box is well ventilated or maintained with an air cooling system.

Caution :
To prevent fire or shock hazard, UL listed wire VW-1, style 1007, should be used for the cable.

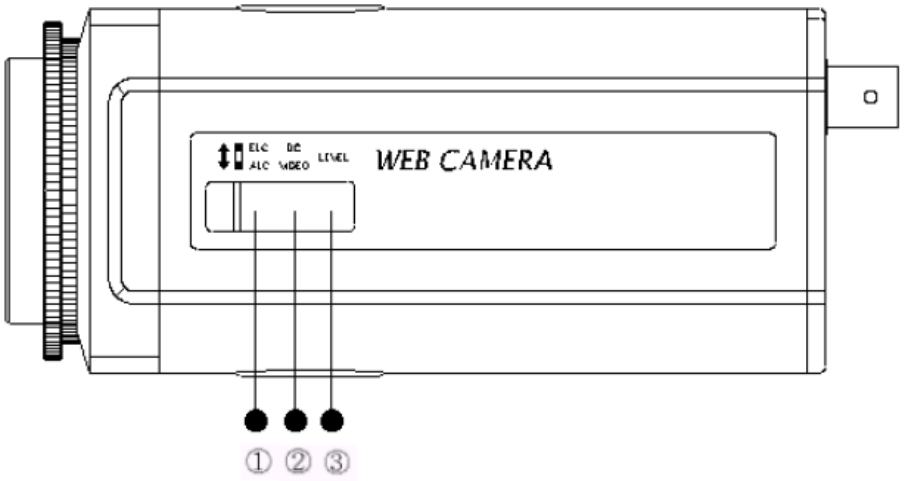
1. WEB CAMERA/VIDEO SERVER Preview

WEB CAMERA/VIDEO SERVER is the set top box type equipment, which transmits real-time digital video data through Internet network.
For operating WEB CAMERA/VIDEO SERVER, it's unnecessary an additional equipment or program, just need power and network cable connecting. You can see the camera images through web browsers (MS INTERNET EXPLORER, NETSCAPE COMMUNICATOR) at anytime, in anywhere.

2. Component Names and Function(WEB CAMERA)

Familiarize yourself with the names and function of each part on the side and rear panels before installing WEB CAMERA.

Side View of WEB CAMERA



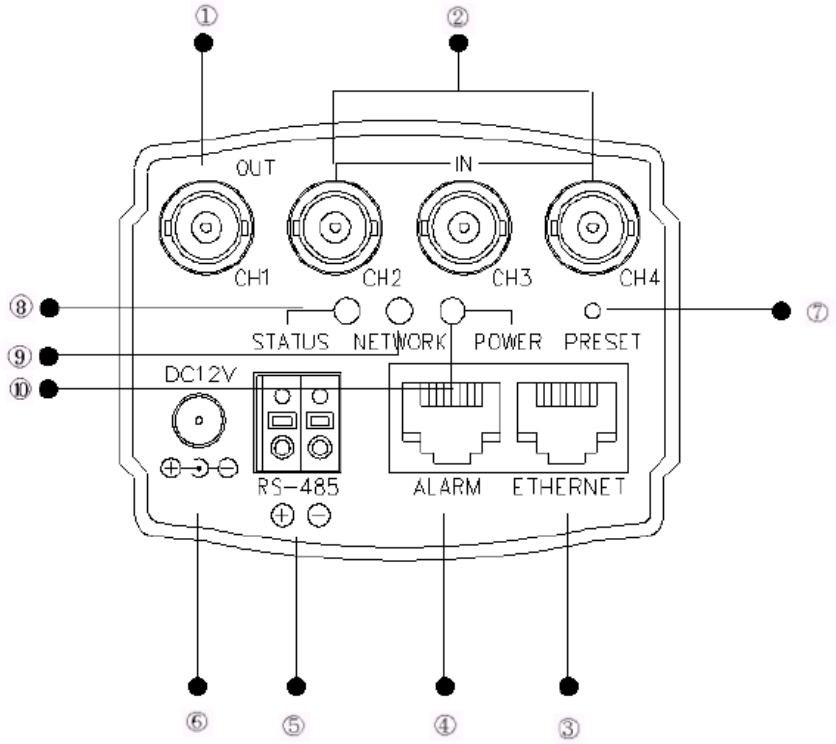
Name	Function
① ELC/ALC	UP : ELC(Electronic Light Control) Mode Down : ALC(Automatic Light Control) Mode In the ELC mode, a continuously variable electronic shutter is employed to automatically control the exposure time of the CCD image sensor according to the incoming light level. With this mode selected, a fixed or manual iris lens can be used instead of an auto IRIS lens. In the ALC mode, the CCD shutter speed is fixed to 1/60(1/50)sec, and the incoming light level is controlled by the auto IRIS lens.
② DC/VIDEO	UP : DC Drive Lens Down : VIDEO Drive Lens To use a video-drive type or the DC-drive type of auto IRIS Lens, set ELC/ALC switch to the ALC position.

WEB CAMERA/VIDEO SERVER

③ LEVEL	ALC level adjustment potentiometer for DC drive lens. This potentiometer is used only if the camera is fitted with a DC-drive auto IRIS lens. It is used to control the amount of light level.

Monitor screen	LEVEL control direction
To increase the brightness	clockwise
To decrease the brightness	counterclockwise

Rear View of WEB CAMERA

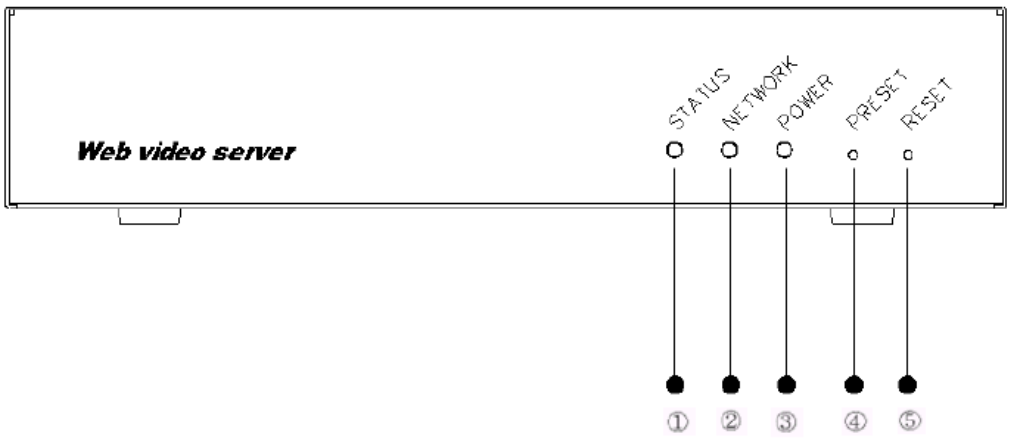


WEB CAMERA/VIDEO SERVER

Name		Function
①	VIDEO OUT	Video out connector of WEB CAMERA.
②	VIDEO IN	Extra Video input connector.
③	ETHERNET	It is the network connecting port for Ethernet (10Base-T/100Base-T).
④	ALARM	It is terminals which could input and output alarm. Use the cable supplied. Users may select N.O and N.C according to be installed equipment.
⑤	RS-485	In order to control PTZ through web browser at remote place, connect WEB CAMERA to PTZ driver with RS-485 cable
⑥	DC 12V	Power input connector. It' s accepts 12V DC power source.
⑦	PRESET	Used when setting figures need to be returned to the initial set-up environment of factory default. Used when forgot the password to access Administration or need to return all setting figures to the initial set-up environment.
⑧	STATUS	Indicate connecting status of network cable. It is ON when LAN cable is connected normally.
⑨	NETWORK	Flickering when transmitting and receiving data after connecting network normally.
⑩	POWER	Indicate the power ON/OFF

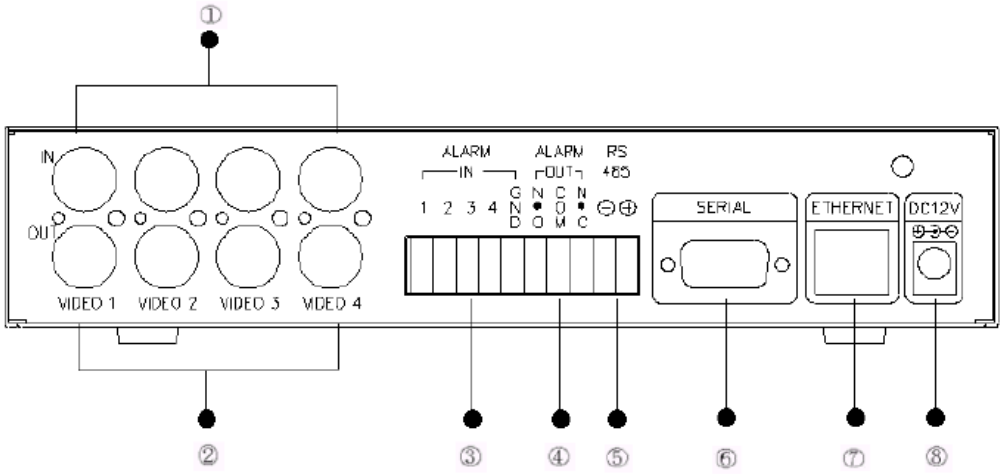
3. Component Names and Function(WEB VIDEO SERVER)

Front View of WEB VIDEO SERVER



Name	Function
① STATUS	Indicate connecting status of network cable. It is ON when LAN cable is connected normally.
② NETWORK	Flickering when transmitting and receiving data after connecting network normally.
③ POWER	Indicate power ON/OFF status.
④ PRESET	Used when setting figures need to be returned to the initial set-up environment of factory default. Used when forgot the password to access Administration or need to return all setting figures to the initial set-up environment.
⑤ RESET	Used when re-operating WEB VIDEO SERVER.

Rear View of WEB VIDEO SERVER



Name	Function
① VIDEO IN	Camera input connector.
② VIDEO OUT (Loop Through Out)	In case of supplying camera signal to outside equipment, possible to use by connecting to this terminal. When connecting, automatically 75Ω termination resistor off.
③ ALARM IN	It is the terminals which could input alarm sensors.
④ ALARM OUT	Possible to select N.O (Normal Open) / N.C (Normal Close) type.
⑤ RS-485	In order to control PTZ through web browser at remote place, connect WEB VIDEO SERVER to PTZ driver with RS-485 cable.
⑥ SERIAL	It is the serial port for external modem.
⑦ ETHERNET	It is the network connecting port for Ethernet (10Base-T/100Base-T).
⑧ DC-12V	Power input connector. It' s accepts 12V DC power source.

4. System Connections

- Power Connection
  - Supply power to WEB CAMERA/VIDEO SERVER as DC-12V input terminal.

- Video Input Connection
  - Connect camera to this input terminal in the rear side of WEB CAMERA/VIDEO SERVER.

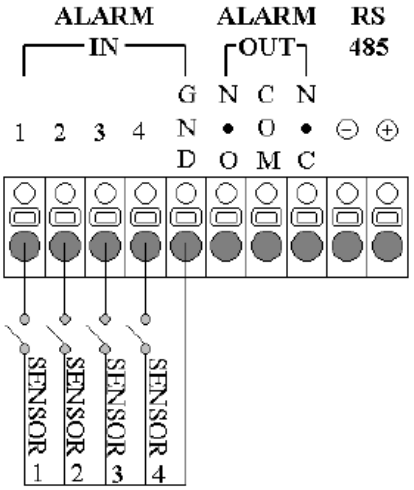
- Video Output Connection
  - Connect monitor or any other CCTV equipment to this terminal in the rear side of WEB CAMERA.

- Network Connection
  - To assign an IP first time, connect LAN cable which is provided with WEB CAMERA/VIDEO SERVER. After setting, please use normal direct LAN cable when install WEB CAMERA/VIDEO SERVER to the place where you want.

- Alarm Cable Connection (Only WEB CAMERA)
  - Connect both alarm input and relay output to alarm terminal located a backside of WEB CAMERA/VIDEO SERVER depend on speciality. Connect alarm equipment to " ALARM OUT" terminal in rear-side of WEB CAMERA/VIDEO SERVER. Possible to use by selecting N.O (Normal Open) and N.C (Normal Close).

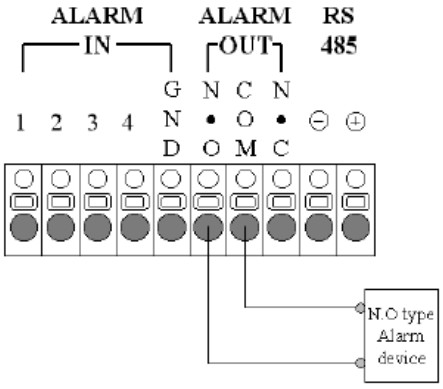
	ALARM CABLE LINE COLOR	CONNECTION
Alarm Input	White-Orange	GND
	Orange	ALM-1
	White-Green	ALM-2
	Blue	ALM-3
	White-Blue	ALM-4
Alarm Output	Green	COM
	White-Brown	N.C
	Brown	N.O

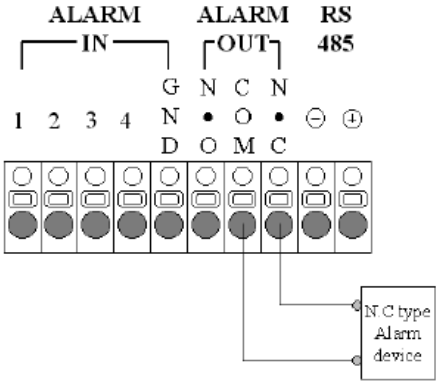
• Alarm Input Connection (Only WEB VIDEO SERVER)



Connect output of sensor, alarm input terminal in rear-side according to allocated channel numbers. Connect one side of sensor to allocated numbers, the other side to contact terminals indicated " GND " .

• Alarm output connection (Only WEB VIDEO SERVER)



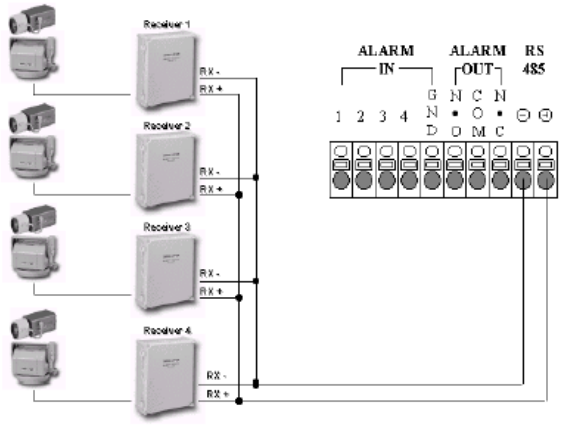


<N.O(Normal Open Type) Connection>

<N.C(Normal Close Type) Connection>

Connect alarm equipment to " ALARM OUT" terminal in rear-side of WEB VIDEO SERVER. Possible to use by selecting N.O (Normal Open) and N.C (Normal Close).

- RS-485 Connection

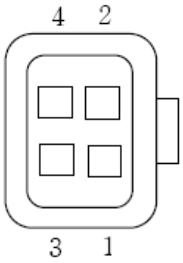


Connect the RS-485 input of the Receiver to RS-485 output in rear-side of WEB CAMERA/VIDEO SERVER.
When connecting, be careful of polarity.
Connect " + " terminal in rear-side of WEB CAMERA/VIDEO SERVER to " + " terminal of receiver RS-485 Input, Connect " - " terminal of RS-485 output to " - " terminal of receiver RS-485 Input. In case of connecting polarity in wrong way, remote control of Pan/Tilt/Zoom could not work.

- Serial Connection (Only WEB VIDEO SERVER)

In ISDN and PPP networks, the Serial port is used as the network port after connection with WEB VIDEO SERVER.

- Auto IRIS lens connector for DC drive lens & VIDEO drive lens (Only WEB CAMERA)



Side of Camera

Pin \ Lens	Video-Drive Lens	DC-Drive Lens
1	POWER SUPPLY(9V DC,40mA)	Damp(-)
2	N.C	Damp(+)
3	VIDEO OUTPUT(07Vp-p without sync)	Drive(+)
4	GROUND	Drive(-)

5. Setting IP Address

No information upon network set up except Ethernet (MAC address) in factory defaults. You must set up the IP address when installing.

• Before you begin

- To use WEB CAMERA/VIDEO SERVER under Network environment, Public IP address must be assigned to WEB CAMERA/VIDEO SERVER from a network manager. At this time, check the following network related information.

  ✓ Gateway
  ✓ Subnet mask
  ✓ DNS

- MAC address is recorded on the bottom of WEB CAMERA/VIDEO SERVER at the time of purchase. Because it may be accidentally erased, please duplicate the number and keep it somewhere else. Check MAC address which is written in the bottom of WEB CAMERA/VIDEO SERVER. You must keep this number for IP address set-up.

- You must only use public IP address, when setting up IP address in Ethernet. If you use private IP address, it's available to connect them within intranet.

5-1. Setting IP by connecting to LAN Cable

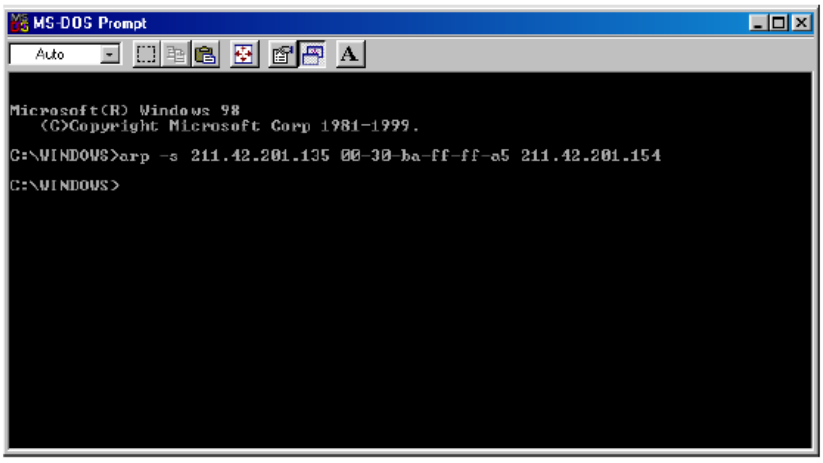
Install the application program " j2re-1_3_1-win-I" that is Java program in accessory CD.
Connect WEB CAMERA/VIDEO SERVER and network connected PC using the LAN cable supplied.

* Setting IP using " arp -s" command in MS-DOS

1. Open " MS-DOS Prompt" window from network-connected PC with WEB CAMERA/VIDEO SERVER.

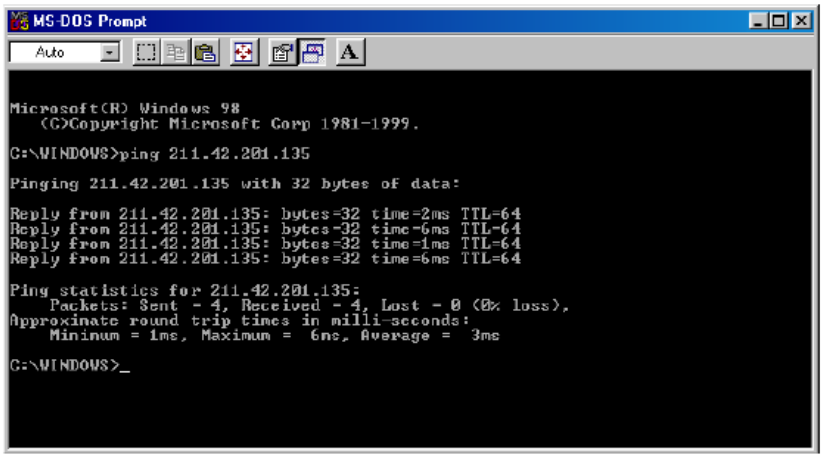
2. Execute " arp -s"

- arp -s < WEB CAMERA/VIDEO SERVER IP Address> < WEB CAMERA/VIDEO SERVER MAC address> <PC IP address>
- At this time, WEB CAMERA/VIDEO SERVER's IP and PC's IP of upper class must equal.
- " arp - s" command must be executed in 20 minutes after power supply.

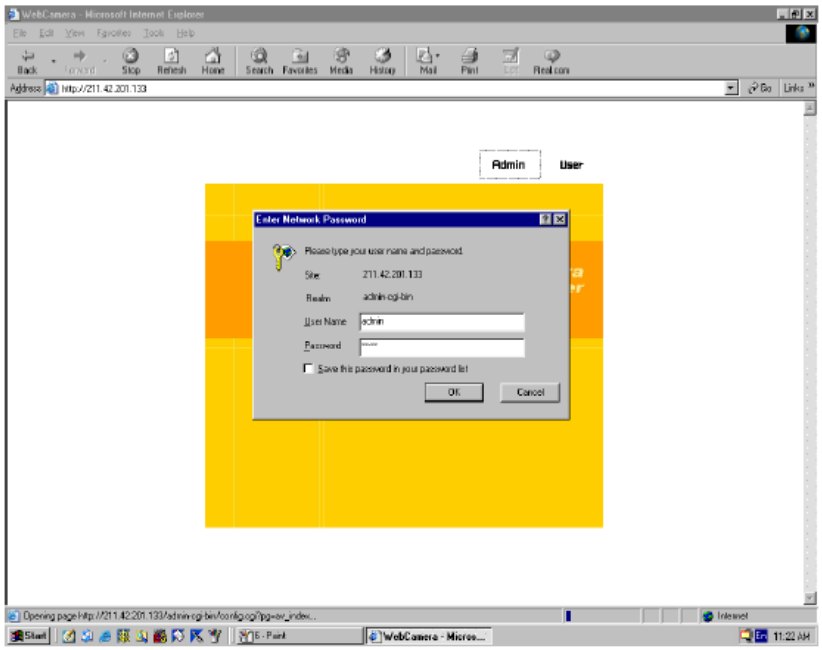


3. Execute " ping"

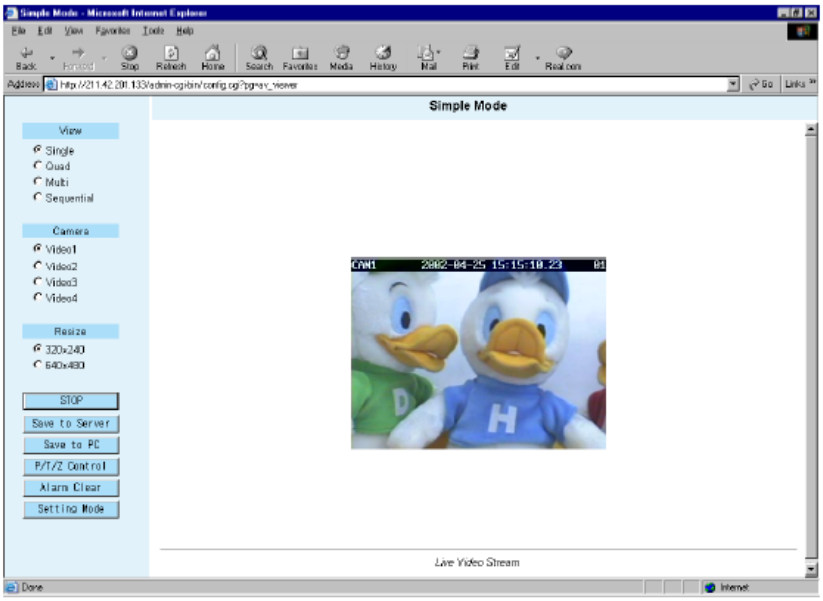
- ping < WEB CAMERA/VIDEO SERVER IP address>
- If you getting no response, attempt 1 ~ 2 more times.
- In case of no response of " Ping" , " Request times out" message will be shown.



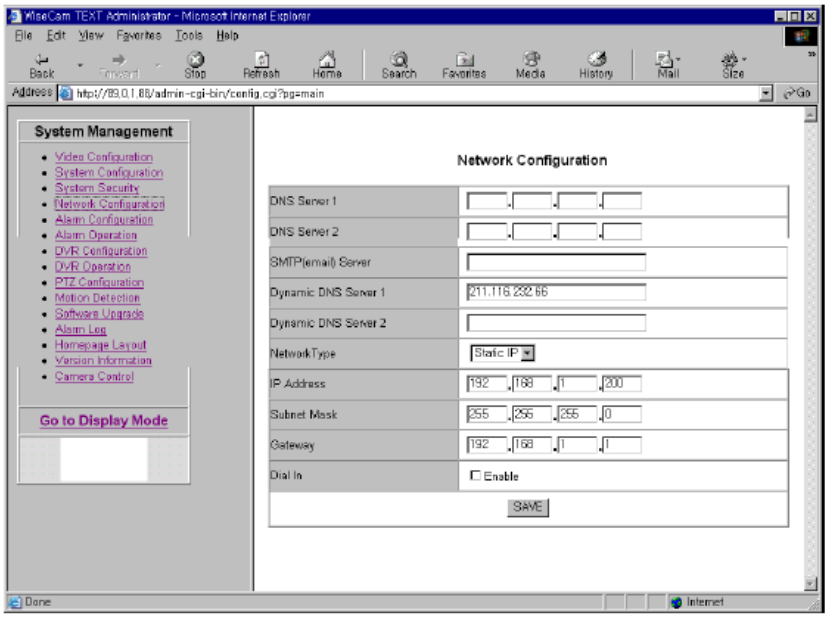
4. Open the web browser, and enter the IP address on the address line and connect to WEB CAMERA/VIDEO SERVER's Admin Mode.
Default value is set as follows :
ID: admin
Password: admin0



5. If you click setting mode in the bottom of left side, it will go to Setting mode. Following screen is the initial screen according to change of simple mode to Setting mode.



6. Select " Network Configuration" menu of " System Management" in Admin Mode.



Enter the WEB CAMERA/VIDEO SERVER's IP address and detailed information regarding the IP and push the " SAVE" button.

7. Finally, click " Reboot" button after choosing " System Configuration" of " System Management" menu in the Admin Mode.
Following screen will be displayed.

WEB CAMERA/VIDEO SERVER

When you click " OK" , WEB CAMERA/WEB SERVER will restart.
It takes 30 ~ 60 sec for restarting.

8. If you connect web camera again, you will find that new set up applied.

5-2. Setting IP by connecting to Serial Cable

(Only WEB VIDEO SERVER)

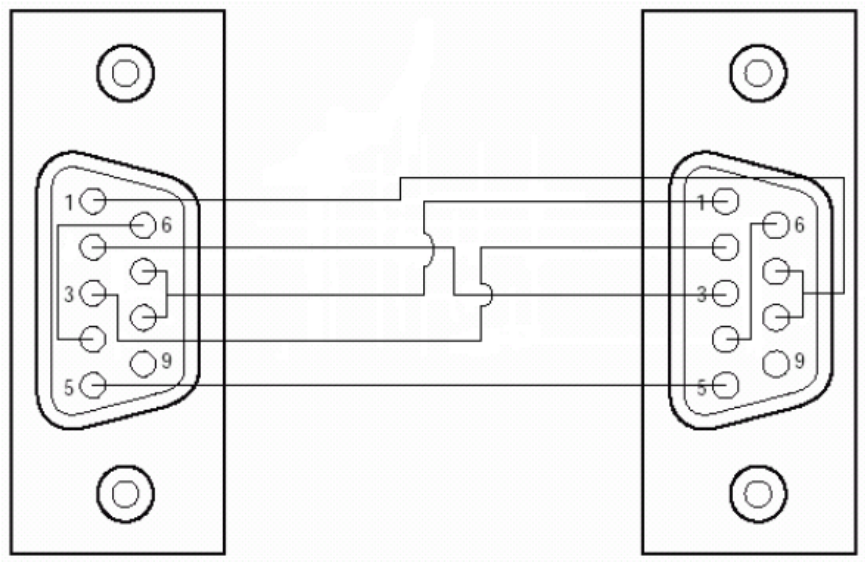
- Before you begin

This method makes easy to set various menus of initial network setting using the RS-232C cable. It could be used from PC which has MS Window Operation System.

The type of RS-232C cable should be used same as below. Serial COM(common) port could be used with telephone modem or mouth so that you should check if other devices in system registration information are using serial communication port.

- Making RS-232C Cable

RS-232C Cable does not include in the WEB CAMERA/VIDEO SERVER so that you should produce additionally. Please refer to below diagram.



< 9 Pin Female to 9 Pin Female >

- Setting terminal emulation communication

WEB CAMERA/VIDEO SERVER

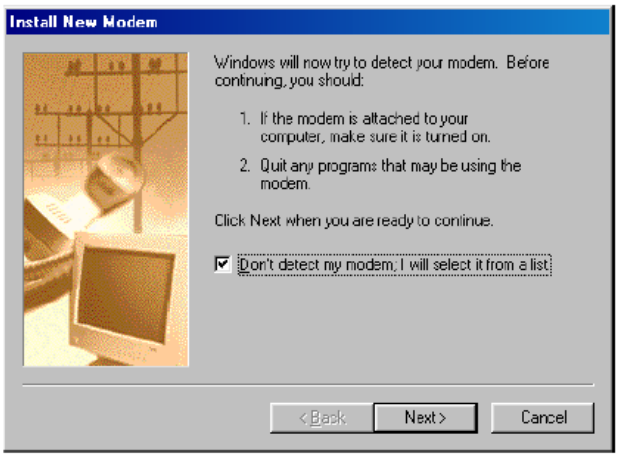
Item	Setting condition
Baud rate	152000 bps
Data bit	8 bit
Stop bit	1 bit
Parity	None
Flow Control	None

* Setting connecting program

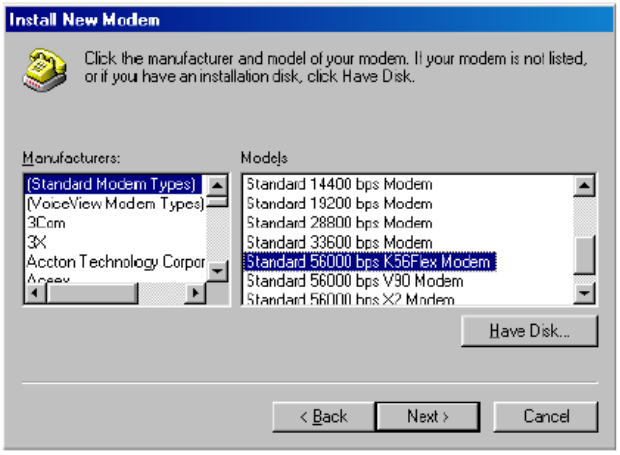
1. After connecting serial communication port of WEB CAMERA/VIDEO SERVER with serial communication port of PC to connect though RS-232C cable, please connect to power source.

2. In the " Start" menu of Window, move to " Settings" and choose " Control Panel" . Double click the " Modem" button.

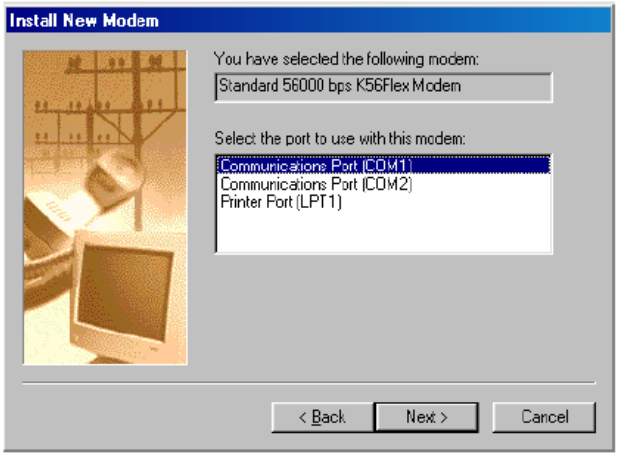
3. Please check the " Don' t detect my modem ; I will select it from a list" and then click " Next " .



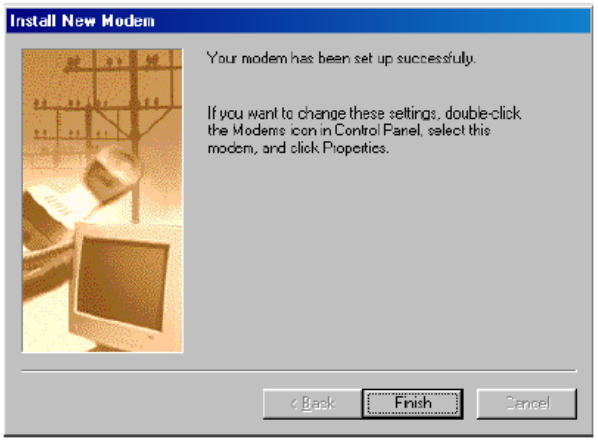
4. Choose the " Modem" to install. And choose " Standard 56000bps K56Flex Modem" basically as below, and then click " Next" in order to continue.

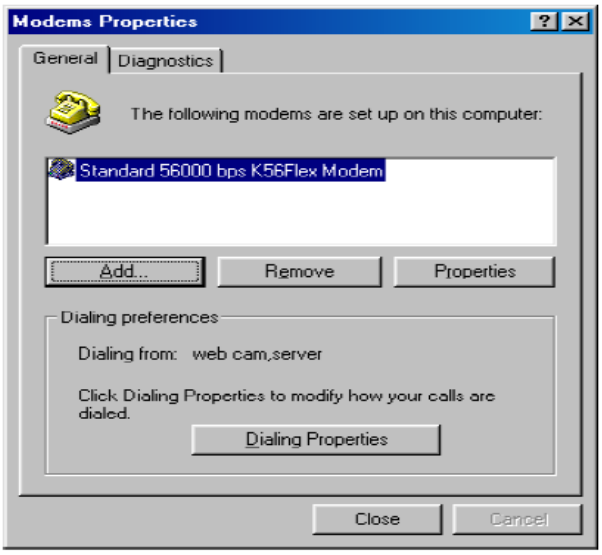


5. After selecting the port to use with this modem, click " Next" .

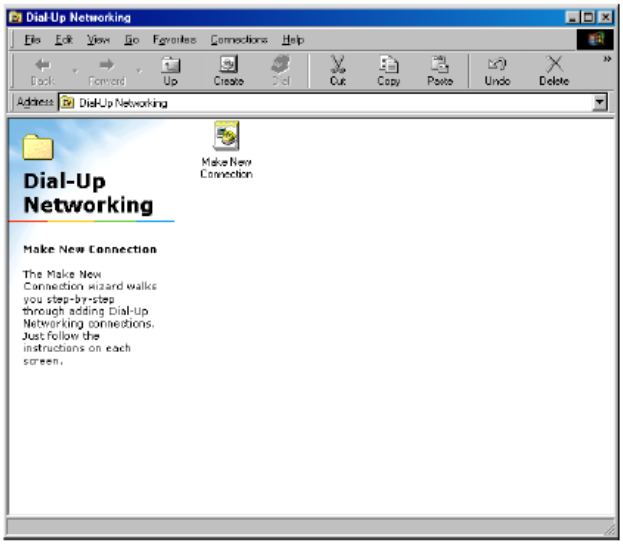


6. After checking if installation status is in correct way, click " Finish" button. And click the " Close" button.

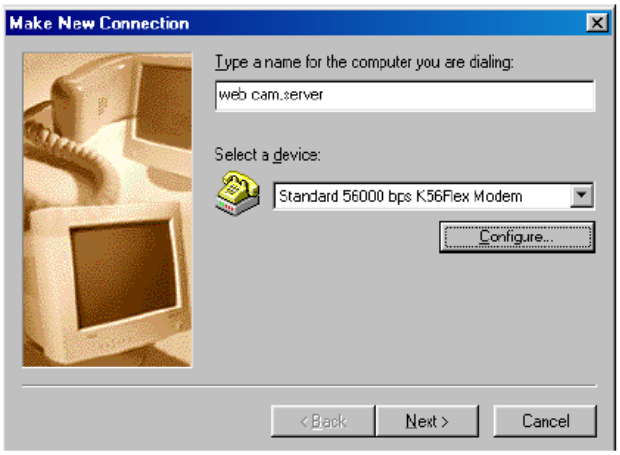


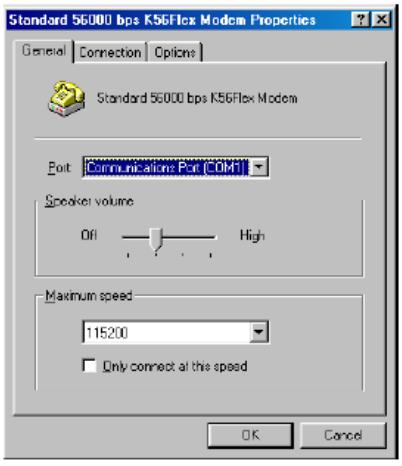


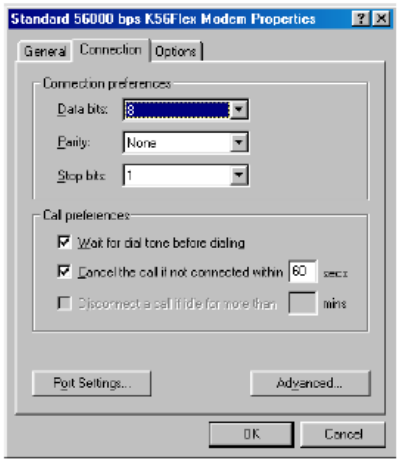
7. In the " Start" menu of the Window, move from " Programs" , " Accessories" to " Communications " and choose " Dial-up Networking" and then double click " Make New Connection" as below.



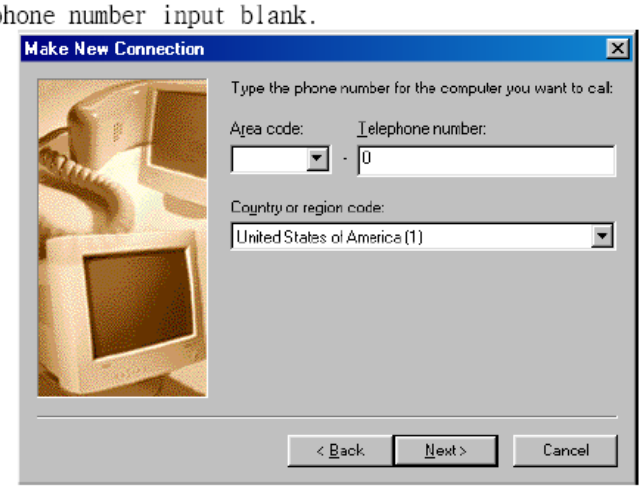
8. After typing name of setting connection, click " Next" button. And Check the " Configure" items as blows.



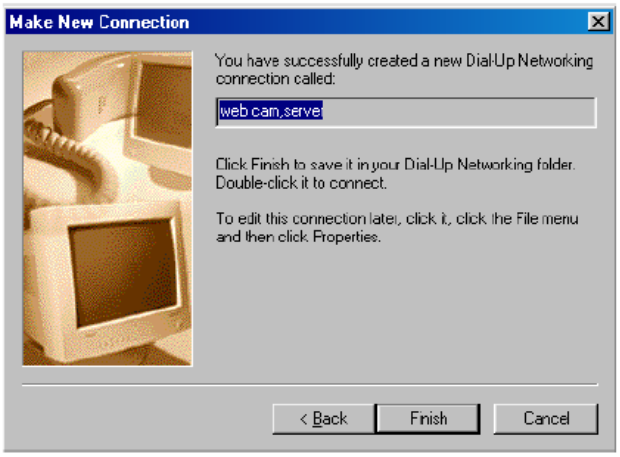




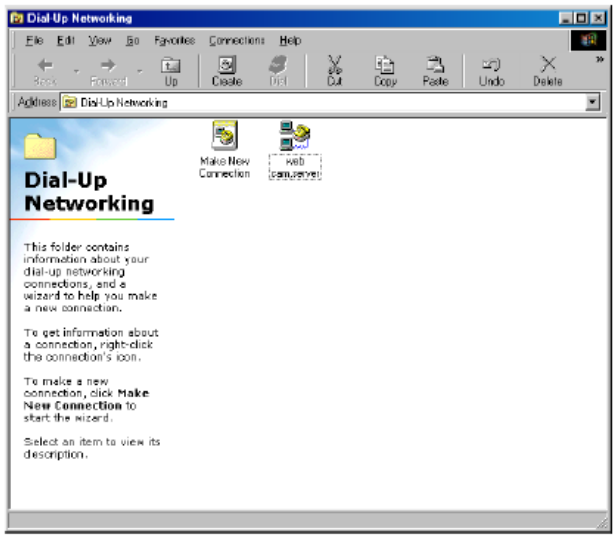
9. Input telephone number to connect. At this time, input optional number " 0" in the phone number input blank.



10. If you complete " Dial-up Networking", finish setting by clicking " Finish" button.



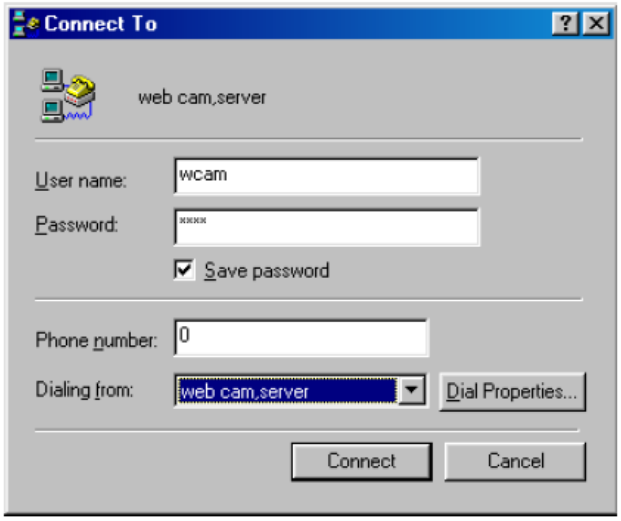
11. After checking if a new icon is produced, check setting status by clicking " Properties".



12. Start Connecting by clicking appropriate icon. In the initial connection, below screen shows and initial user name and password of WEB CAMERA/VIDEO SERVER are as below.

User Name : wcam

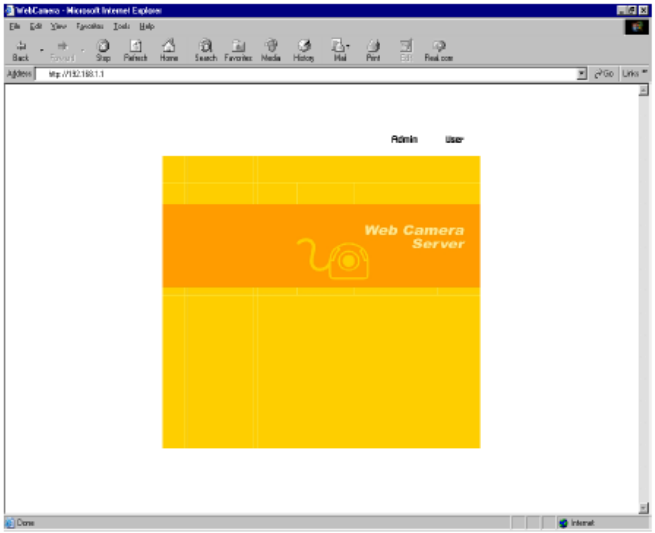
Password : wcam



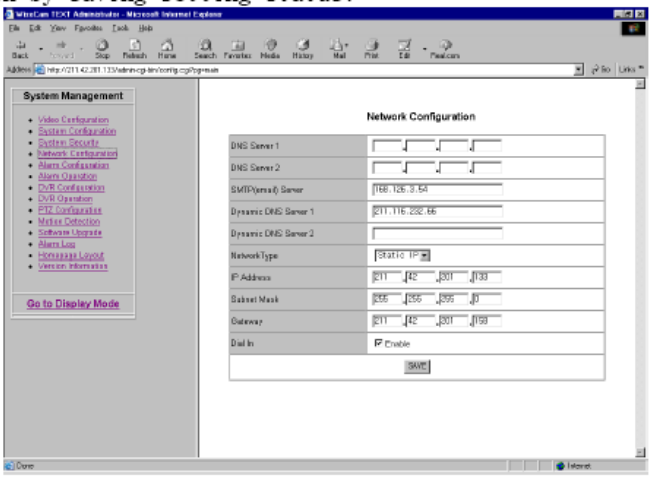
13. Check the message as below which is informing connection status is normal.

14. Execute MS Internet Explorer and connect to Admin Mode after inputting exclusive use of serial telecommunication IP Address, " 192.168.1.1" , into Address Window as below.

WEB CAMERA/VIDEO SERVER



15. In the " System Management" menu of Admin Mode, choose " Network Configuration", input IP Address and Network information and click " SAVE" button by saving setting status.



16. Finally, click " Reboot", " OK" button after choosing " System Configuration" of " System Management" menu in the Admin Mode.

6. Log - in

After successfully setting up an IP address, please install WEB CAMERA/VIDEO SERVER where you want. At that time, please use Direct LAN Cable not Cross LAN Cable supplied.
Direct LAN Cable is not supplied. As for Direct LAN Cable, please contact Network Manager.

When you finished camera in a proper place, please execute a web browser (Netscape or Explorer) from a PC.

- Recommended web browser
  -. MS Internet Explorer 5.0 or above
  -. Netscape communicator 4.7x

- User/Admin Name and password at the time of purchase is defaulted as follows

1) User Mode
   User Name : user
   Password : user

2) Admin Mode
   Admin Name : admin
   Password : admin0

6-1. User Mode Log-in

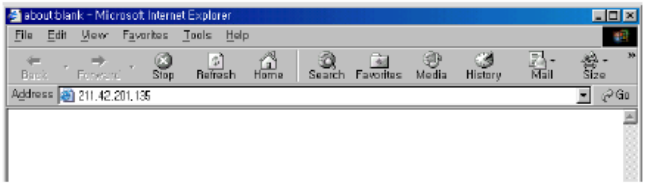
User Mode allows the user to only view displayed images without altering settings.
This applies when a user enters " user" for both user name and password or the network administrator has assigned a user name and password for logging in purposes
The process to log-in in User Mode varies according to the web browser.

6-1-1. Using MS Internet Explorer

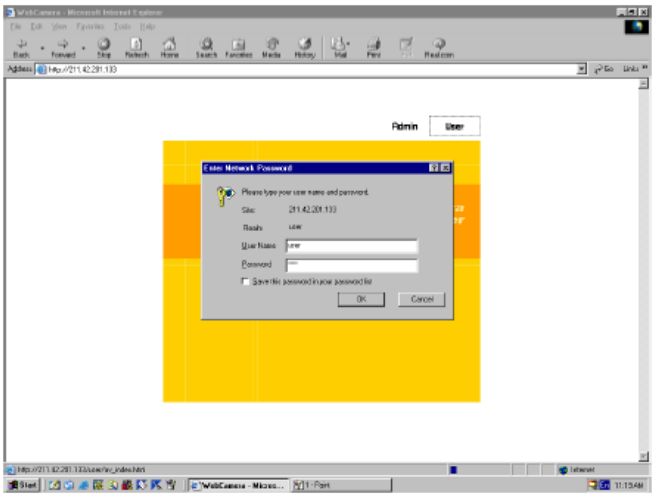
If you never logged in through explorer, first download ActiveX Install from WEB CAMERA/VIDEO SERVER. Please follow next procedure.

1. Open the " MS Internet Explorer" web browser.

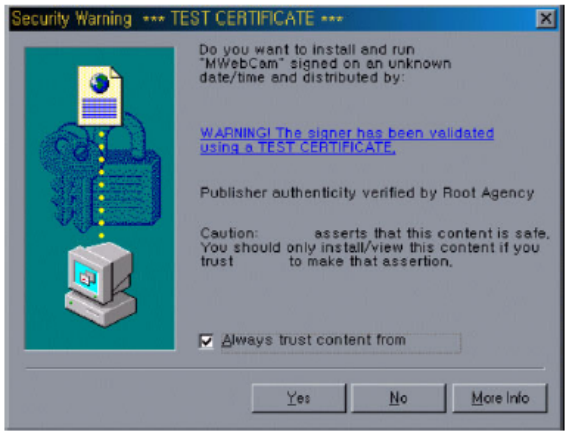
2. Enter the WEB CAMERA/VIDEO SERVER IP address in go to : xxx.xxx.xxx.xxx



3. Click the " User" icon, and the following Log-in window appears.



4. Properly enter user name and password and click " OK" icon. Then, you should see the security message.



5. Click " Yes" . You just have completed downloading Explorer ActiveX Install from WEB CAMERA/VIDEO SERVER. Then, you will see the initial screen.

6. Refer to " Chapter 5. User mode" for a description of the User Mode.

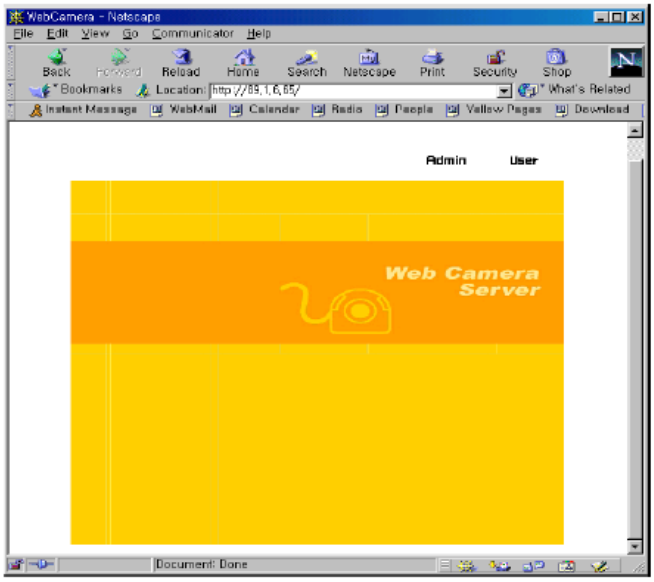
7. After the initial connection, authenticating the ActiveX program is not necessary.

6-1-2. Using Netscape Communicator

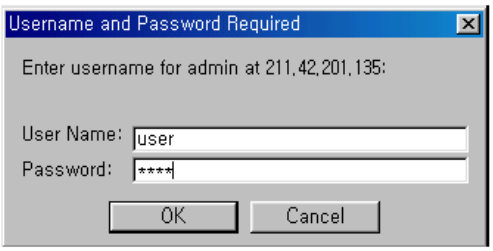
Netscape does not require a special confirmation.

1. Open Netscape web browser, Enter the WEB CAMERA/VIDEO SERVER IP address in go to : xxx.xxx.xxx.xxx

2. If you have correctly logged in, you will see the followings.



3. If you are “User”, please click “User”. You will see the following windows.



4. Enter Admin/user name and password and click “OK” icon. You should see the initial screen of Admin/User mode. Refer to “Chapter 5. User mode” for a description of the User Mode.

6-2. Admin Mode Log-in

WEB CAMERA/VIDEO SERVER can be changed and managed in Admin Mode.
Java 1.3 version should be installed to PC to connect to WEB CAMERA/VIDEO SERVER.
Java 1.3 version is included in accessory CD that is offered with WEB CAMERA/VIDEO SERVER.

To connect in Admin Mode, the process is the same for both MS Internet Explorer and Netscape.

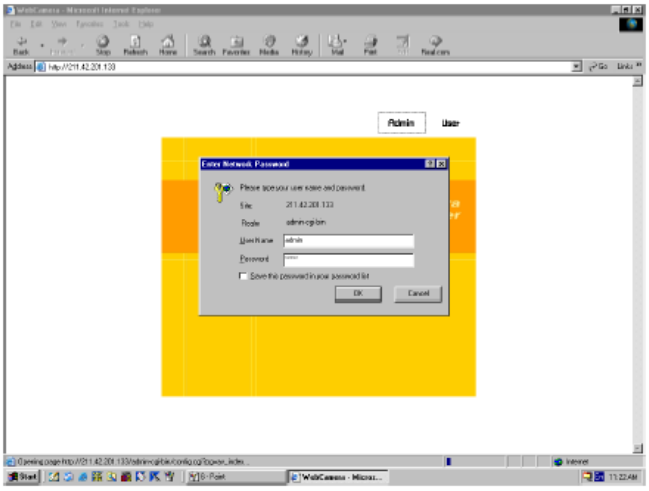
1. The most recent version of the Java Program (Java 1.3 Version) needs to be installed in the PC before WEB CAMERA/VIDEO SERVER can be installed. Computers with Java 1.3 do not require this process.

2. Open Internet Explorer on the PC being used.

3. Enter the WEB CAMERA/VIDEO SERVER IP Address in the address bar and press " Enter."

4. After successfully logging on to WEB CAMERA/VIDEO SERVER, the following Log-In window appears.

5. Enter user name and password and click " Enter," and the first screen for WEB CAMERA/VIDEO SERVER Admin Mode appears. At the time of purchase, the Admin Mode user name and password are programmed as " admin."



6. Refer to “ Chapter 6. Admin Mode” for a description of the Admin Mode menu.

The detailed “ Admin Mode” setting refers User Manual in accessory CD.

Para más información sobre el servidor de vídeo, consultar el manual completo o la hoja de especificaciones. Ambos documentos están disponibles para su descarga en:

http://www.linudix.com/eng/product/lws700_s1.html

5. Protocolo TCP/IP

Debido a la extensión de este documento (80 páginas) no se incluirá en la versión impresa del proyecto. Si se desea, puede consultarse en el CD adjunto o en la dirección siguiente:

http://www.rfc-es.org/rfc/rfc0793-es.txt

6. Protocolo FTP

Debido a la extensión de este documento (70 páginas) no se incluirá en la versión impresa del proyecto. Si se desea, puede consultarse en el CD adjunto (en formato PDF) o en la dirección siguiente (Tanto el contenido de la Web como el archivo están en inglés):

http://tools.ietf.org/html/rfc959

6. Especificación de C#

Debido a la extensión de este documento (519 páginas) no se incluirá en la versión impresa del proyecto. Si se desea, puede consultarse en el CD adjunto (en formato DOC) o en la dirección siguiente (en inglés):

http://download.microsoft.com/download/3/8/8/388e7205-bc10-4226-b2a8-75351c669b09/CSharp%20Language%20Specification.doc

Pamplona, Septiembre 2010

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

Elaboración de una interfaz de comunicación para módulos domóticos

PRESUPUESTO

Asier Ortés Alfonso

Tutor: Ignacio Raúl Matías Maestro

Pamplona, Septiembre de 2010

Índice

1. Herramientas Software 289
2. Material 290
3. Mano de obra 291
4. Total 292

1. Herramientas Software

Concepto	*Descripción*	*Unidades*	*Precio Unitario (€)*	*Total (€)*
Microsoft Office 2010 Hogar y pequeña empresa	Aplicaciones de uso general. Incluye Word, Excel, Access y Powerpoint.	1	379,00	379,00
Microsoft Visual Studio 2008 Proffesional Edition	Entorno de programación para C#. Herramienta fundamental que permite la compilación y ejecución de los programas.	1	800,00	800,00
E.Commander	Herramienta software del fabricante para el manejo de los módulos.	1	580,00	580,00
Gastos de envío (UPS Standard)	-	1	13,39	13,39
TOTAL	**-**	**-**	**-**	**1772,39**

2. Material

Concepto	*Descripción*	*Unidades*	*Precio Unitario (€)*	*Total (€)*
E.Reader	Módulo 8 entradas analógicas, 6 entradas/salidas digitales, 2 relés	1	1850,00	1850,00
e.bloxx A1-8	Módulo 8 entradas analógicas, 8 entradas analógicas y 8 salidas digitales	1	1407,00	1407,00
e.bloxx D2-1	Módulo 4 relés	1	340,00	340,00
ISK200	Convertidor RS232	1	280,00	280,00
ICM100	Conector de bus	4	7,00	28,00
IBT100	Terminación de bus	2	7,00	14,00
ICL100	Conector ISK200 y e.reader	1	15,00	15,00
TSL 120	Fuente de alimentación	1	118,00	118,00
DES-1008D	Hub	1	57,86	57,86
LWS-700	Servidor de video IP	1	217,00	217,00
THR-370/CM	Sensor de humedad relativa y temperatura	1	231,00	231,00
Gems Sensor 0-250	Sensor de depresión	1	131,72	131,72
CDM4161	Sensor de CO2	1	90,01	90,01
PT-100	Sensor de temperatura	2	43,55	87,10
TOTAL	**-**	-	-	**4866,69**

3. Mano de obra

Concepto	*Unidades*	*Precio Unitario (€)*	*Total (€)*
Documentación y estudio	400	20,00	8000,00
Programación de la aplicación	200	30,00	6000,00
TOTAL	**-**	**-**	**14000,00**

4. Total

Concepto	*Unidades*	*Precio Unitario (€)*	*Total (€)*
Herramientas Software	1	1772,39	1772,39
Material	1	4866,69	4866,69
Mano de obra	1	14000,00	14000,00
Suma Conceptos Anteriores	**-**	**-**	**20639,08**
IVA (18%)	**-**	**-**	**3715,03**
TOTAL	**-**	**-**	**24354,11**

El total del presupuesto asciende a la cantidad de **VEINTICUATRO MIL TRESCIENTOS CINCUENTA Y CUATRO EUROS CON ONCE CÉNTIMOS**.

Firma del autor:

Pamplona, Septiembre 2010

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

Elaboración de una interfaz de comunicación para módulos domóticos

BIBLIOGRAFÍA

Asier Ortés Alfonso

Tutor: Ignacio Raúl Matías Maestro

Pamplona, Septiembre de 2010

1. “Beginning C# 2008: From Novice to Professional”. Christian Gross. Ed. Apress.
2. “Illustrated C# 2008”. Daniel Solis. Ed. Apress.
3. “C# Essentials”. Ben Albahari, Peter Drayton, Brad Merrill. Ed. O’Reilly.
4. “C# 3.0 in a nutshell”. Joseph Albahari, Ben Albahari. Ed. O’Reilly.
5. “Programming C# 3.0”. Jesse Liberty, Donald Xie. Ed. O’Reilly.
6. “.NET Windows forms in a nutshell”. Ian Griffiths, Matthew Adams. Ed. O’Reilly
7. “Windows forms programming in C#”. Chris Sells. Ed. Addison-Wesley.
8. “The C# programming language”. Anders Hejlsberg, Scott Wiltamuth, Peter Golde. Ed. Addison-Wesley.
9. http://www.gantner-instruments.com/
10. http://msdn.microsoft.com/es-es/library/ms123401.aspx
11. http://msdn.microsoft.com/es-es/vcsharp/dd919145.aspx
12. http://www.elguille.info/
13. http://www.wikipedia.org/
14. http://www.codeproject.com/
15. http://www.zedgraph.org/
16. http://www.forosdelweb.com/
17. http://www.netdeveloper.com
18. http://www.devjoker.com/
19. http://www.programacionfacil.com/

Pamplona, Septiembre de 2010

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

Elaboración de una interfaz de comunicación para módulos domóticos

ANEXO I: MANUAL DE USUARIO

Asier Ortés Alfonso

Tutor: Ignacio Raúl Matías Maestro

Pamplona, Septiembre de 2010

Índice

1 Introducción 3

1.1 Descripción del software 3
1.2 Requisitos del sistema 3

2 Descripción de la interfaz 4

2.1 Bienvenida 4
2.2 Menú principal 5
2.3 Configuración 6
2.4 Captura 8
2.5 Gráficos 10
2.6 Video 11
2.7 Ayuda 12

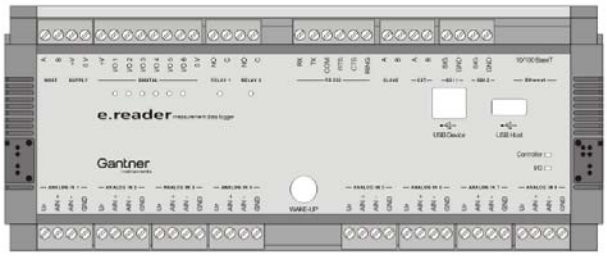
3 Primeros Pasos 13

3.1 Instalación 13
3.2 Configuración 13
3.3 Captura 13
3.4 Representación gráfica 13
3.5 Video 14

1. Introducción

1.1 Descripción del software

Este software ha sido diseñado para su uso con módulos de tipo e.reader de la marca Gantner Instruments. El principal cometido del programa es ofrecer una interfaz de usuario interactiva y sencilla para el control y uso de dichos módulos.



Módulo E.reader de Gantner Instruments

Una vez se dispone de una instalación basada en un módulo E.reader y una serie de sensores conectados a él (ya sea de forma directa o mediante módulos de expansión compatibles). Este programa ofrece la posibilidad de configurar los parámetros más importantes de cada entrada, por lo que la instalación adquiere flexibilidad. Asimismo, este software permite visualizar los valores capturados por los sensores tanto en tiempo real como de manera histórica, lo que brinda un control total de la instalación. Además de esto, el programa también soporta la conexión con un servidor de video IP con seguridad habilitada, lo que facilita aún más si cabe el control remoto de la instalación con la seguridad de una contraseña para que solo los usuarios autorizados pueda acceder al contenido.

1.2 Requisitos del sistema

El software ha de instalarse en un PC con Windows 98 o superior. El espacio libre de memoria en disco requerido por el programa son 90MB. También se ha de tener en cuenta el espacio de almacenaje de los datos que puede llegar a 200MB por año (tomando muestras cada 10 minutos). El PC deberá contar con un conector tipo RJ-45 (ethernet) así como con una tarjeta de red.

2. Descripción de la interfaz

Una vez copiados los archivos del programa al PC se puede proceder a ejecutar la aplicación.

2.1 Bienvenida

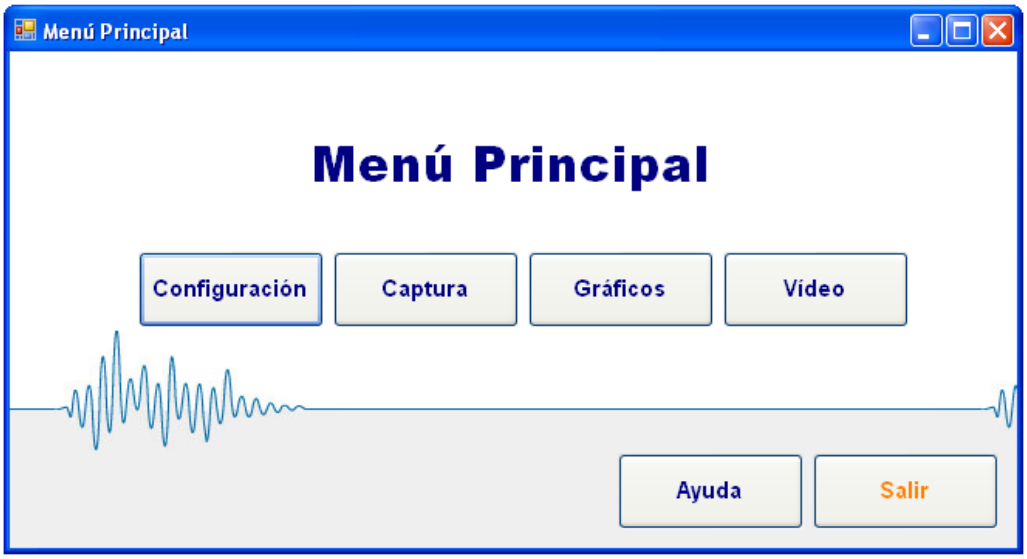
Al cargar el programa aparecerá la pantalla de bienvenida. Es el punto de inicio y final del programa, por lo que tanto al acceder como al cerrar la aplicación se habrá de pasar por él. Consta de dos botones: Entrar y Salir. Pulsando Entrar se accede al Menú principal de la aplicación.

2.2 Menú principal

El menú principal es la ventana desde la que se accede a todas las funciones del programa. Consta de dos filas de botones. En la fila superior se encuentran los botones que dirigen a partes activas del programa, es decir, a las diferentes funcionalidades, a saber: Configuración, Captura, Gráficos y Vídeo. En la ventana de configuración se podrán realizar los cambios en la configuración del e.reader y los módulos de expansión que haya conectados. En la ventana de captura, se podrán visualizar los valores instantáneos que entreguen los sensores de la instalación. La ventana de gráficos muestra los valores capturados en una gráfica dependiente del tiempo. Por último, la ventana de vídeo vincula la aplicación con el servidor de video.

La fila inferior cuenta con dos botones. Pulsando el botón situado más a la izquierda se accederá a la ventana de ayuda, donde pueden encontrarse las directrices básicas de manejo del programa. El botón de la derecha redirige al usuario a la ventana de bienvenida. Como puede observarse, este botón contiene letras de color anaranjado, distintas a las de los demás. Esto indica el camino de salida óptimo para la aplicación. Siempre que se encuentre un botón con letras de este color, se sabrá que dirige a la ventana anterior a la que se encuentre. EN este caso es la ventana de bienvenida.

En la primera utilización se recomienda leer el contenido de la ventana de ayuda y posteriormente acceder al apartado de configuración, como se explicará en el apartado reservado a los primeros pasos.



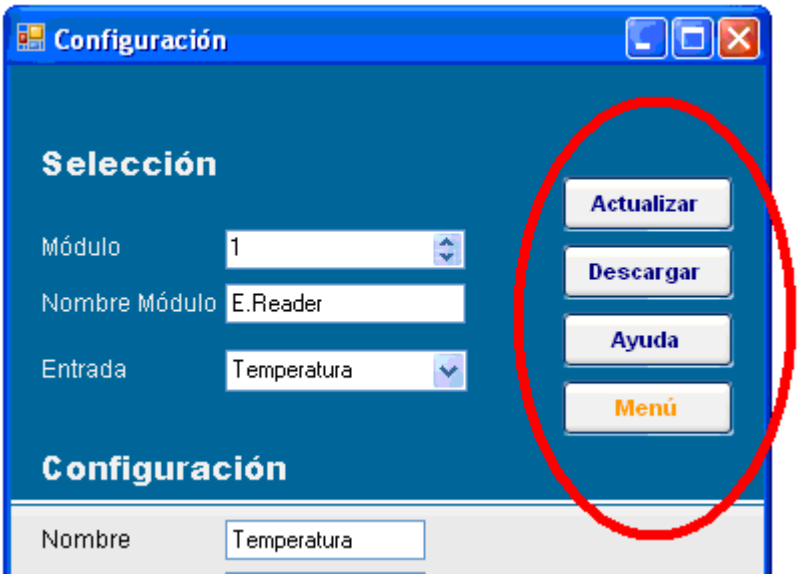
2.3 Configuración

La ventana de configuración es la más compleja de la aplicación. En ella pueden modificarse los parámetros que afectan a cada una de las entradas del e.reader y de los módulos de expansión asociados a él. En el apartado de Selección se elegirá la entrada de la cual se quiere visualizar la configuración, seleccionando previamente el módulo al que pertenece.

En el apartado de configuración se pueden observar los parámetros que es posible modificar, entre los que se encuentran el nombre, el lugar en el que se ubica el módulo, el tipo de entrada o salida, las unidades del sensor conectado a la entrada, los límites entre los que se desea que se encuentre el valor a medir (previsión para implementar alarmas o conectar actuadotes al sistema) o la precisión (número de decimales) con la que se presentarán los valores que se capturen.

La sección de información adicional muestra un par de datos técnicos, que pueden ser de utilidad si se desea conectar sensores más complejos. En principio se reserva este espacio para información personalizada dependiendo de la actividad para la que desee usarse.

Pulsando los botones de la parte superior derecha de la pantalla la aplicación se comunica con los módulos llevando a cabo las siguientes funciones:



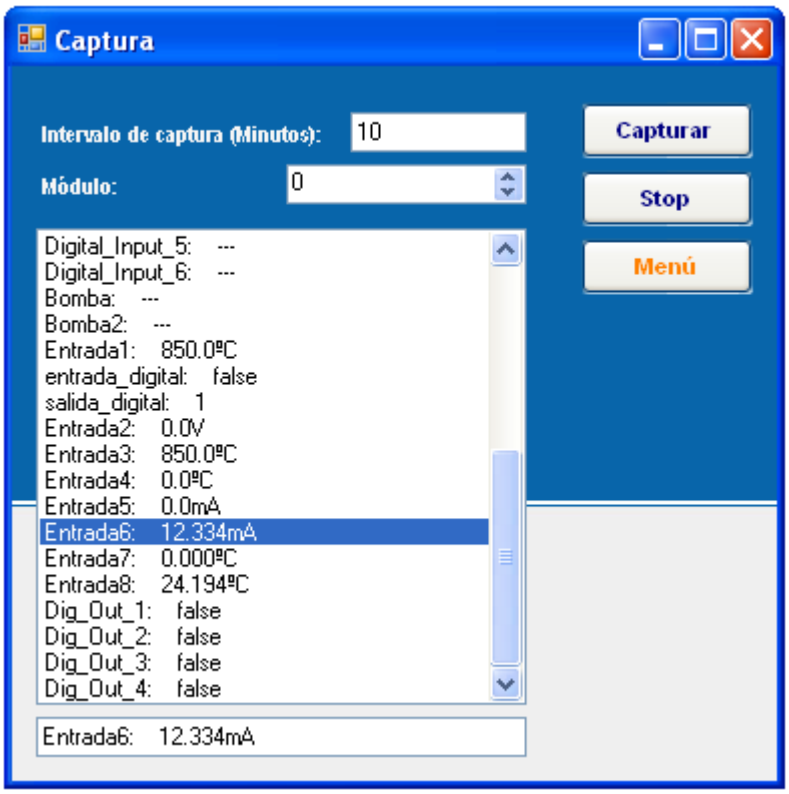
El botón Actualizar guarda los cambios que se hayan hecho a los parámetros y actualiza la configuración del e.reader. Es muy importante desconectar los sensores del módulo al hacer algún cambio importante en la configuración ya que de haber algún error podrían dañarse. Si los cambios son de pequeña índole, como cambios de nombre, de precisión o límites no será necesario desconectar los sensores. El proceso de actualización ha de hacerse entrada por entrada y dura un minuto.

El botón Descargar captura la configuración actual del módulo y la muestra. Esta opción es de gran utilidad la primera vez que conecta el módulo o cuando se desea hacer una comprobación de que una configuración se ha hecho correctamente. Se recomienda hacer una descarga cada vez que se haga una actualización. El proceso de descarga se hace una sola vez para todas las entradas y dura 30 segundos. Cuando se termina el proceso de descarga el contador de módulos se sitúa en la siguiente posición al último módulo encontrado, esto ayuda a conocer el número de módulos activos en la instalación.

El botón de ayuda redirige a la ventana de ayuda, donde se podrá encontrar información útil sobre el uso del programa y como llevar a cabo los procesos más habituales.

2.4 Captura

La ventana de captura es el punto del programa donde se adquieren los datos de los sensores conectados a la instalación. Además de adquirirse los valores, estos pueden ser visualizados prácticamente en tiempo real.



En esta pantalla, se puede elegir el intervalo de captura que se desee (cada cuantos minutos se realizarán las capturas) escribiendo el número de minutos en el cuadro reservado a tal efecto. Con el control *Módulo* se elegirán los valores que se visualizarán, aunque es recomendable dejar su valor en cero ya que de esta manera se mostrarán todas las entradas y no solo las correspondientes al módulo en cuestión.

El botón Capturar inicia la adquisición de los datos, que se repetirá cada 10 minutos si no se indica contrariamente en el cuadro del intervalo de captura. La adquisición de datos es inmediata, por lo que no habrá a penas tiempo de espera salvo en la primera adquisición o en caso de que el número de entradas sea muy elevado. En cualquier caso el tiempo no excederá de unos pocos segundos. Este tiempo irá marcado por un mensaje en el cuadro de texto de la parte inferior de la ventana.

El botón Stop detiene la captura de datos. Para reiniciarla solo se debe volver a pulsar el botón de captura.

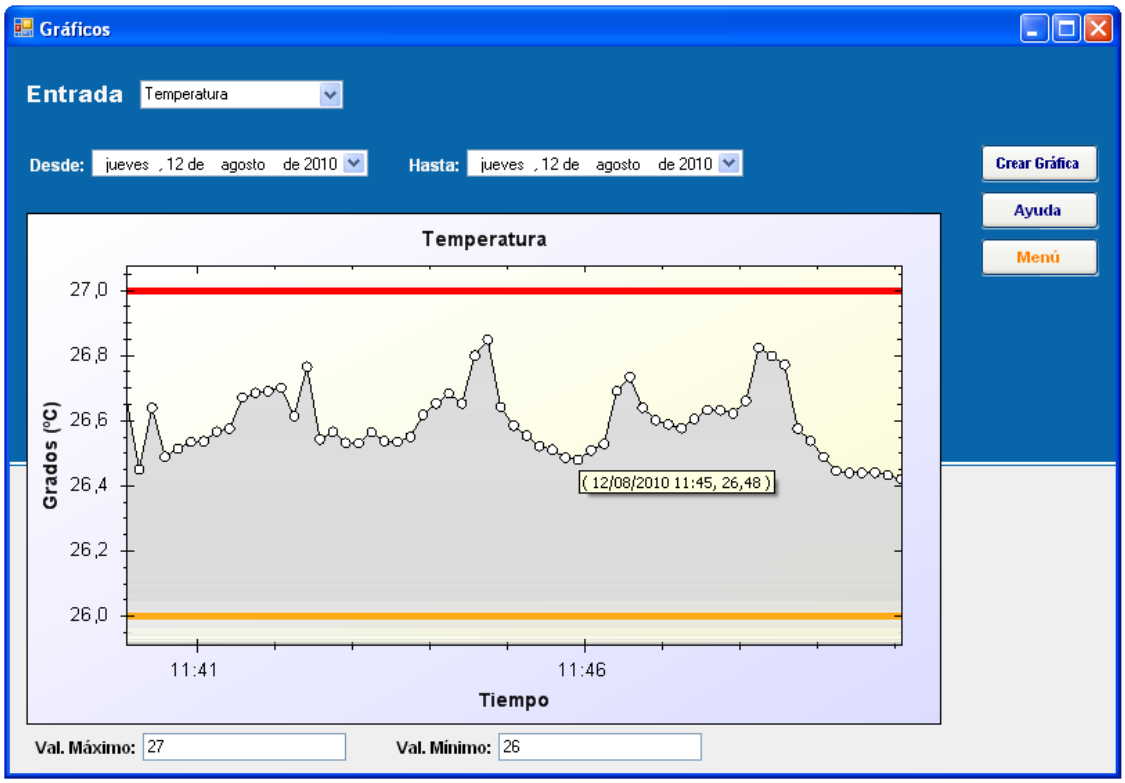
Esta ventana se encontrará habitualmente minimizada y con la captura activa. Dado que el tiempo de adquisición es mínimo, no impide seguir trabajando con el resto de las funcionalidades de la aplicación. De esta manera, se conseguirá tener un seguimiento continuo de los valores de la instalación. El único caso en el que es conveniente detener la captura es cuando se realizan cambios en la configuración, ya que el módulo necesita un tiempo para que la configuración se estabilice y si coincide con una de las capturas podría fallar.

2.5 Gráficos

En la pantalla de gráficos puede accederse a información histórica de la instalación. Consta de un control de tipo lista con el que se elegirá la entrada a visualizar. En un principio el control se encontrará vacío, una vez pulsado el botón de creación de la gráfica aparecerán los nombres de las entradas configuradas.

Los dos selectores de fecha que se encuentran debajo de la lista de entradas se utilizan para indicar el intervalo de días de los que se desea ver la información. El intervalo mínimo es de un día y no existe intervalo máximo.

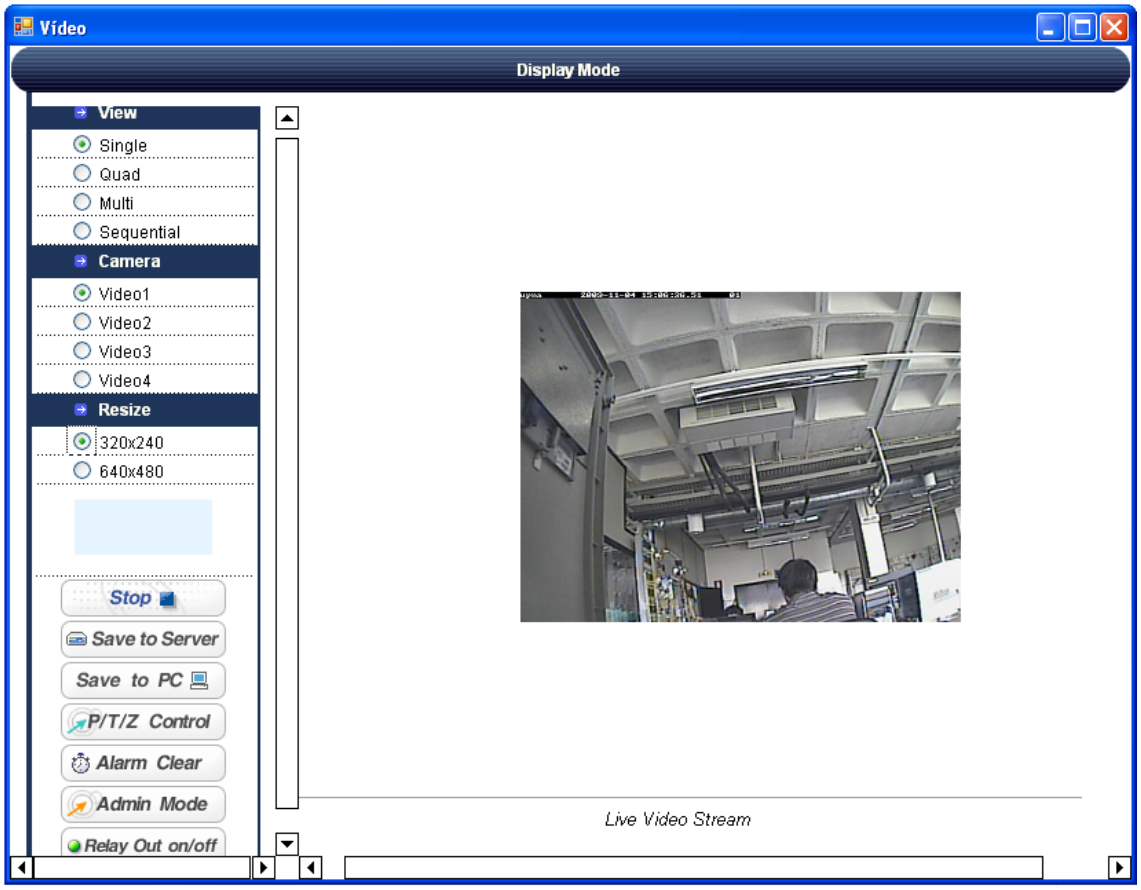
Los dos cuadros de texto rotulados máximo y mínimo en la parte inferior de la ventana contendrán los valores entre los que es deseable que se encuentre el parámetro que se represente. El valor máximo estará representado en la gráfica como una línea gruesa de color rojo y el valor mínimo cambiará el rojo por el naranja.



El botón de creación de gráfica carga la lista de entradas que se pueden visualizar. Una vez elegida se dibuja la curva, que dará como resultado una representación similar a la de la fotografía. El tiempo de dibujo de la curva habitualmente se sitúa entorno a uno o dos segundos. En caso de que el número de valores a representar en la gráfica sea muy elevado (puede ocurrir cuando el intervalo es muy grande o cuando se han realizado un número excepcional de capturas), este tiempo puede aumentar. Se recomienda no superar las 500 capturas para evitar una espera demasiado prolongada.

2.6 Video

En esta ventana se redirige al usuario a la interfaz del servidor de video. Será necesario, en primer lugar, introducir un nombre de usuario y una contraseña válidas. El servidor de video utilizado es el LWS-700 del fabricante Linudix. Para más información consultar el manual de usuario propio del producto. Una vez se introduce la contraseña la interfaz tiene el aspecto siguiente:

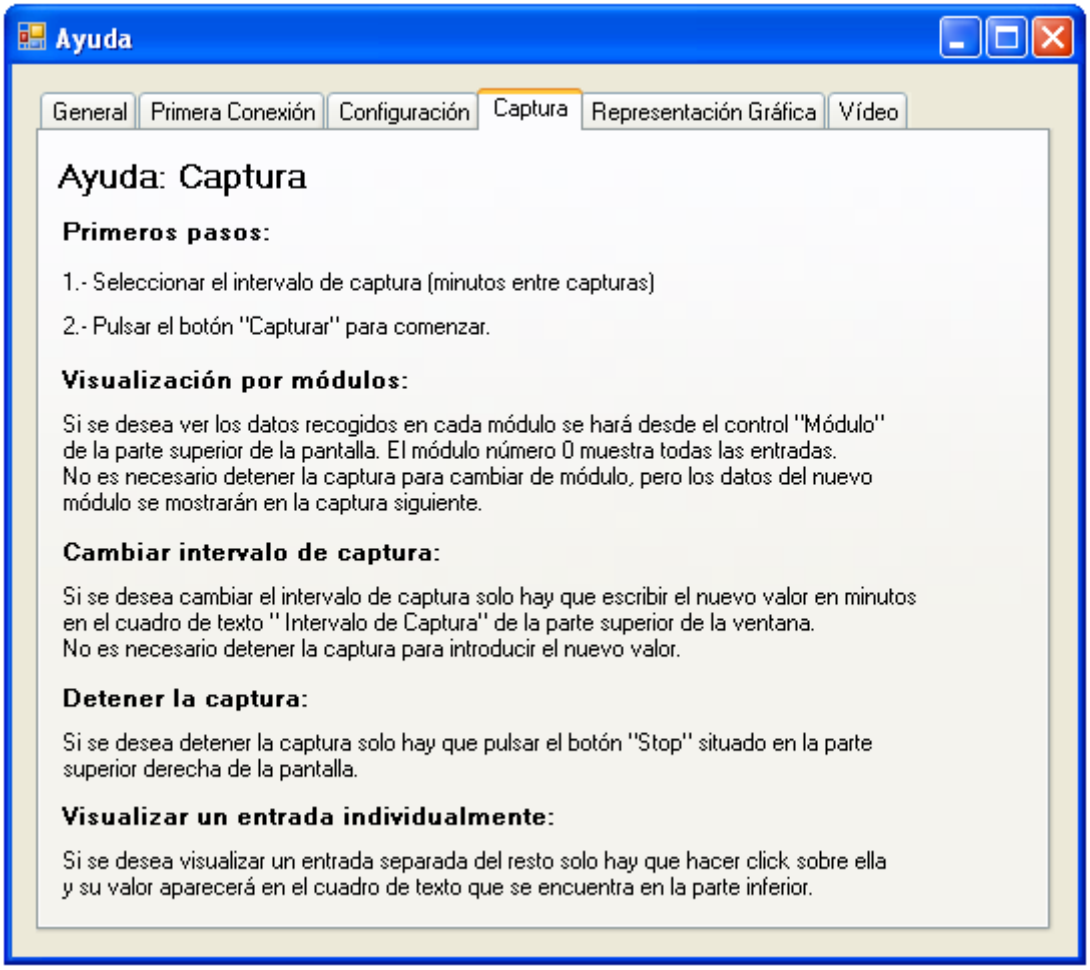


En el menú de la parte izquierda de la ventana se pueden acceder a todas las funcionalidades de la interfaz. Como puede observarse, el servidor tiene capacidad para soportar hasta cuatro cámaras emitiendo simultáneamente, con lo que muchas de las opciones se centran en la combinación de las cámaras en la visualización. También es posible cambiar el tamaño de la imagen de 320x240 píxeles a 640x480.

Además de las opciones de visualización, más a bajo se encuentran el resto de las opciones, entre las que destaca el acceso al modo administrador, desde el que se podrá modificar la clave de acceso, la IP y otros parámetros relacionados con la comunicación y el acceso al servidor.

2.7 Ayuda

La ventana de ayuda contiene la información necesaria para comenzar a utilizar la aplicación desde cero. Está dividida en diferentes pestañas a fin de estructurar su contenido dependiendo de la sección de la aplicación a la que se refiera. En la fotografía siguiente puede verse su aspecto:



Las pestañas en las que está dividida la ventana son: General (en la que se da información sobre las funciones que realiza cada parte del programa), Primera conexión (en la que se especifican las primeras acciones a realizar la primera vez que se utiliza el programa), Configuración (en ella se explican las funciones de configuración), Captura (en la que se explica como utilizar esta ventana), Representación gráfica (se ofrecen breves explicaciones sobre las funciones de esta ventana) y Vídeo (que sigue la misma línea de las demás pestañas).

Todas las pestañas constan de dos partes, una primera en la que se explica como comenzar a utilizar las funciones de la ventana en cuestión y una segunda donde se dan ejemplos de acciones que se pueden realizar. En la fotografía se puede observar la pestaña dedicada a la ventana de captura.

3. Primeros pasos

3.1 Instalación

- Conecte el e.reader y los módulos de expansión entre si y a la fuente de alimentación. (Para más información sobre este punto consultar el manual del e.reader)

- Conecte el e.reader mediante cable cruzado a su PC o mediante cable recto a un switch o hub. En caso de utilizar un hub o switch asegúrese de que su PC está correctamente conectado al él.

- Ponga el e.reader en funcionamiento y espere unos segundos hasta que las luces del dispositivo queden fijas.

3.2 Configuración

- Acceda a la ventana de configuración y pulse el botón “Descargar” para poder visualizar la configuración actual del módulo.

- Cambie los parámetros de cada entrada que desee utilizar y actualícelas.

- Conecte los sensores a las entradas correspondientes y observe que todas las luces sean de color verde. Si alguna de ellas tiene color rojo, asegúrese de que el sensor está bien conectado. Si este nos es el problema, desconéctelo y vuelva a configurar la entrada.

- Vuelva a pulsar el botón descargar y compruebe que la actualización se ha llevado a cabo correctamente. Esto pondrá de manifiesto cualquier problema que se haya dado con la configuración.

3.3 Captura

- Cierre la ventana anterior y acceda a la ventana de captura.

- Pulse el botón “Capturar” y analice los valores recogidos por los sensores. Si observa alguna anomalía detenga la captura y vuelva a configurar las entradas afectadas. Repita la operación hasta que los valores sean correctos.

3.4 Representación gráfica

- Una vez se hayan realizado varias capturas minimice la ventana de captura y abra el menú principal.

- Acceda a la ventana de representación gráfica.

- Seleccione la fecha actual en los cuadros Desde y Hasta.

- Pulse el botón Crear Gráfica y seleccione una entrada de la lista.

- Una vez elegida la entrada haga clic sobre el cuadro de la gráfica para que esta se muestre.

3.5 Video

- Acceda a la ventana de video pulsando en el botón correspondiente del menú principal.

- Pulse el botón “Display Mode” para entrar en el modo visualización.

- Escriba el nombre de usuario y contraseña que se le habrán proporcionado con el servidor de video y pulse aceptar.

Para obtener más información sobre como realizar las operaciones más frecuentes del programa consulte la ventana de Ayuda.

Pamplona, Septiembre de 2010

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

Elaboración de una interfaz de comunicación para módulos domóticos

ANEXO II: CÓDIGO DE LA APLICACIÓN

Asier Ortés Alfonso

Tutor: Ignacio Raúl Matías Maestro

Pamplona, Septiembre de 2010

Índice

1 Pantalla de bienvenida 3
2 Menú principal 4
3 Pantalla de configuración 5
4 Pantalla captura 13
5 Pantalla gráficos 16
6 Biblioteca de clases y punto de entrada 20

1. Pantalla de bienvenida

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Form3 Form3 = new Form3();
            Form3.Visible = true;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            this.Close();
        }


    }
}
```

2. Menú principal

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form3 : Form
    {
        FTPFactory ff = new FTPFactory();
        public Form3()
        {
            InitializeComponent();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            Form2 Form2 = new Form2();
            Form2.Visible = true;
            this.Close();
        }

        private void button4_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Form4 Form4 = new Form4();
            Form4.Visible = true;
            this.Close();

        }

        private void button3_Click(object sender, EventArgs e)
        {
            Form5 Form5 = new Form5();
            Form5.Visible = true;
            this.Close();
        }

        private void button5_Click(object sender, EventArgs e)
        {
            Form6 Form6 = new Form6();
            Form6.Visible = true;
        }

        private void button6_Click(object sender, EventArgs e)
        {
            Form7 Form7 = new Form7();
            Form7.Visible = true;
        }
    }
}
```

3. Pantalla de configuración

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Xml;

namespace WindowsFormsApplication1
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }
        string eltexto = "";
        string eltexto2 = "";
        int i = 0;
        int cont = 0;
        int k = 0;
        String[] Nom = new String[1000];

        private void numericUpDown1_ValueChanged(object sender, EventArgs e)
        {
            int i = 0;
            //cargamos el número de archivo en i
            i = decimal.ToInt32(numericUpDown1.Value);
            //borramos los nombres de las anteriores entradas de la combobox
            this.comboBox1.Items.Clear();
            this.comboBox1.Text = "";
            //instanciamos la clase guardarnombres
            guardarnombres NOMBRE = new guardarnombres();
            //instanciamos el objeto nombres y le enviamos el numero de archivo y la
palabra clave
            //Este nos devuelve un string con los
            //nombres de las entradas separados por una barra
            eltexto = NOMBRE.nombres(i, "Na");
            //dividimos el texto que nos devuelve la funcion
            string[] split = eltexto.Split(new Char[] { '/' });
            //creamos un string[]
            string[] Nombres;
            Nombres = new string[split.Length];
            //vamos guardando el texto dividido en el nuevo string
            for (k = 1; k < split.Length; k++)
            {
                Nombres[k] = split[k - 1];
            }
            //cargamos el nuevo string palabra por palabra en el combobox
            for (k = 1; k < split.Length; k++)
            {
                this.comboBox1.Items.Add(Nombres[k]);
                this.Controls.Add(this.comboBox1);
            }
            //NOMBRE DEL MÓDULO
            guardarnombres nombremodulo = new guardarnombres();
            eltexto2 = nombremodulo.nombres(i, "ModTy");
            if (eltexto2 == "230/")
            {
                textBox3.Text = "E.Reader";
            }
            if (eltexto2 == "200/")
```

```
            {
                int j=0;
                j=i-1;
                textBox3.Text = "E.Bloxx-" + j;
            }
            if (eltexto2 == "216/")
            {
                textBox3.Text = "E.Bloxx-Digital";
            }
            if (eltexto2 == "")
            {
                textBox3.Text = "N/A";
            }
            eltexto2 = "";

        }

        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {

            //UNIDADES

            i = decimal.ToInt32(numericUpDown1.Value);
            guardarnombres unidades = new guardarnombres();
            eltexto2 = unidades.reconocimiento(i, "DaUn");
            textBox2.Text = comboBox1.Text;
            int k = 0;
            int x = 0;
            if (eltexto2 != "")
            {
                string[] split = eltexto2.Split(new Char[] { '/', ':', '=', '<', '>',
});
                string[] Unidades;
                Unidades = new string[split.Length];
                for (k = 0; k < split.Length; k++)
                {
                    Unidades[k] = split[k];
                }
                eltexto2 = "";
                x = comboBox1.SelectedIndex;
                if (Unidades[x] == "V")
                {
                    comboBox2.SelectedIndex = 1;
                }
                if (Unidades[x] == "_V")
                {
                    comboBox2.SelectedIndex = 1;
                }
                if (Unidades[x] == "€")
                {
                    comboBox2.SelectedIndex = 0;
                }
                if (Unidades[x] == "_€")
                {
                    comboBox2.SelectedIndex = 0;
                }
                if (Unidades[x] == "_mA")
                {
                    comboBox2.SelectedIndex = 2;
                }
                if (Unidades[x] == "")
                {
                    comboBox2.SelectedIndex = 3;
                }


                //TIPO DE ENTRADA

                guardarnombres tipo = new guardarnombres();
                eltexto2 = tipo.reconocimiento(i, "Ty");
                int k2 = 0;
                string[] split2 = eltexto2.Split(new Char[] { '/', ':', '=', '<', '>',
});
```

```
                string[] Tipo;
                Tipo = new string[split2.Length];
                for (k2 = 0; k2 < split2.Length; k2++)
                {
                    Tipo[k2] = split2[k2];
                }
                eltexto2 = "";

                if (Tipo[x] == "1")
                {
                    comboBox3.SelectedIndex = 0;
                }
                else
                {
                    if (Tipo[x] == "4")
                    {
                        comboBox3.SelectedIndex = 1;
                    }
                    else
                    {
                        if (Tipo[x] == "3")
                        {
                            comboBox3.SelectedIndex = 2;
                        }
                        else
                        {
                            comboBox3.SelectedIndex = 3;
                        }
                    }
                }

                //MÁXIMOS Y MÍNIMOS

                if (Unidades[x] == "___")
                {
                    Unidades[x] = "";
                }
                if (Unidades[x] != "")
                {
                    guardarnombres maximo = new guardarnombres();
                    eltexto2 = maximo.reconocimiento(i, "MaxY");
                    int k3 = 0;
                    string[] split3 = eltexto2.Split(new Char[] { '/', ':', '=', '<',
'>', });
                    string[] Max;
                    Max = new string[split3.Length];
                    for (k3 = 0; k3 < split3.Length; k3++)
                    {
                        Max[k3] = split3[k3];
                    }
                    eltexto2 = "";
                    textBox4.Text = Max[x];

                    guardarnombres minimo = new guardarnombres();
                    eltexto2 = minimo.reconocimiento(i, "MinY");
                    int k4 = 0;
                    string[] split4 = eltexto2.Split(new Char[] { '/', ':', '=', '<',
'>', });
                    string[] Min;
                    Min = new string[split4.Length];
                    for (k4 = 0; k4 < split4.Length; k4++)
                    {
                        Min[k4] = split4[k4];
                    }
                    eltexto2 = "";
                    textBox5.Text = Min[x];

                }
                else
                {
                    textBox4.Text = "N/A";
                    textBox5.Text = "N/A";
                }
```

```
                //LOCALIZACIÓN
                guardarnombres localizacion = new guardarnombres();
                eltexto2 = localizacion.reconocimiento(i, "Loc");
                //hay que quitar la barra de separación ya que el valor es único
                eltexto2=eltexto2.Replace("/","");
                textBox8.Text = eltexto2;
                eltexto2 = "";

                //Fecha

                guardarnombres fecha = new guardarnombres();
                eltexto2 = fecha.reconocimiento(i, "Date");
                int k8 = 0;

                //TIPO DE DATO

                guardarnombres tipodato = new guardarnombres();
                eltexto2 = tipodato.reconocimiento(i, "DaTy");
                int k5 = 0;
                string[] split5 = eltexto2.Split(new Char[] { '/', ':', '=', '<', '>',
});
                string[] Tipoda;
                Tipoda = new string[split5.Length];
                for (k5 = 0; k5 < split5.Length; k5++)
                {
                    Tipoda[k5] = split5[k5];
                }
                eltexto2 = "";
                if (Tipoda[x] == "1")
                {
                    textBox11.Text = "BOOLEAN";
                }
                else
                {
                    if (Tipoda[x] == "7")
                    {
                        textBox11.Text = "UNSINT";
                    }
                    else
                    {
                        if (Tipoda[x] == "8")
                        {
                            textBox11.Text = "FLOAT";
                        }
                        else
                        {
                            textBox11.Text = "N/A";
                        }
                    }
                }

                //DIGITOS ANTES DE COMA

                guardarnombres digaco = new guardarnombres();
                eltexto2 = digaco.reconocimiento(i, "DaFl");
                int k6 = 0;
                string[] split6 = eltexto2.Split(new Char[] { '/', ':', '=', '<', '>',
});
                string[] precision1;
                precision1 = new string[split6.Length];
                for (k6 = 0; k6 < split6.Length; k6++)
                {
                    precision1[k6] = split6[k6];
                }
                eltexto2 = "";

                XmlDocument xmldocfecha = new XmlDocument();
                xmldocfecha.Load("entconfig.xml");
                XmlNodeList fechaact = xmldocfecha.GetElementsByTagName("fecha");
                string fechact = fechaact[0].InnerText;
                textBox10.Text = fechact;
```

```
                //DIGITOS después DE COMA

                guardarnombres digdeco = new guardarnombres();
                eltexto2 = digdeco.reconocimiento(i, "DaPr");
                int k7 = 0;
                string[] split7 = eltexto2.Split(new Char[] { '/', ':', '=', '<', '>',
});
                string[] precision2;
                precision2 = new string[split7.Length];
                for (k7 = 0; k7 < split7.Length; k7++)
                {
                    precision2[k7] = split7[k7];
                }
                eltexto2 = "";

                textBox9.Text = precision2[x];
            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            Form3 Form3 = new Form3();
            Form3.Visible = true;
            this.Close();
        }

        private void button3_Click(object sender, EventArgs e)
        {
            Form6 Form6 = new Form6();
            Form6.Visible = true;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            int nu1val = Convert.ToInt32(numericUpDown1.Value);
            //actualizar el nombre
            guardarnombres actual0 = new guardarnombres();
            actual0.actualizacion(nu1val,comboBox1.SelectedIndex,"Na",textBox2.Text);
            //actualizar la localización
            guardarnombres actual1 = new guardarnombres();
            actual1.actualizacion(nu1val, comboBox1.SelectedIndex, "Loc",
textBox8.Text);
            //actualizar la precisión
            guardarnombres actual4 = new guardarnombres();
            actual4.actualizacion(nu1val, comboBox1.SelectedIndex, "DaPr",
textBox9.Text);
            //actualizar el máximo
            guardarnombres actual5 = new guardarnombres();
            actual5.actualizacion(nu1val, comboBox1.SelectedIndex, "MaxY",
textBox4.Text);
            //actualizar el mínimo
            guardarnombres actual6 = new guardarnombres();
            actual6.actualizacion(nu1val, comboBox1.SelectedIndex, "MinY",
textBox5.Text);
            //actualizar el tipo
            string tipo = "";
            if (comboBox3.SelectedIndex == 0)
            {
                tipo = "1";
            }
            else
            {
                if (comboBox3.SelectedIndex == 1)
                {
                    tipo = "4";
                }
                else
                {
                    if (comboBox3.SelectedIndex == 2)
                    {
                        tipo = "3";
```

```
                }
                else
                {
                    tipo = "";
                }
            }
        }

        guardarnombres actual2 = new guardarnombres();
        actual2.actualizacion(nu1val, comboBox1.SelectedIndex, "Ty", tipo);
        //actualizar las unidades
        string uni = "";
        if (comboBox2.SelectedIndex == 0)
        {
            uni = "ºC";
            //aparte del valor de DaUn hay que cambiar otros parámetros
            string u1 = "2";
            string u2 = "3";
            string u3 = "Pt_100";
            string u4 = "2";
            string x0 = "100";
            string y0 = "0";
            string x1 = "3.90481E2";
            string y1 = "700";
            guardarnombres actualC = new guardarnombres();
            actualC.actualizacion(nu1val, comboBox1.SelectedIndex, "AInMo", u1);
            actualC.actualizacion(nu1val, comboBox1.SelectedIndex, "AInMoPar0", u2);
            actualC.actualizacion(nu1val, comboBox1.SelectedIndex, "SeNa", u3);
            actualC.actualizacion(nu1val, comboBox1.SelectedIndex, "LPCnt", u4);
            actualC.actualizacion(nu1val, comboBox1.SelectedIndex, "X0", x0);
            actualC.actualizacion(nu1val, comboBox1.SelectedIndex, "Y0", y0);
            actualC.actualizacion(nu1val, comboBox1.SelectedIndex, "X1", x1);
            actualC.actualizacion(nu1val, comboBox1.SelectedIndex, "Y1", y1);
        }
        else
        {
            if (comboBox2.SelectedIndex == 1)
            {
                uni = "V";
                string u1 = "0";
                string u2 = "1";
                string u3 = "Voltage";
                string u4 = "2";
                string x0 = "-10";
                string y0 = "-10";
                string x1 = "10";
                string y1 = "10";
                guardarnombres actualV = new guardarnombres();
                actualV.actualizacion(nu1val, comboBox1.SelectedIndex, "AInMo", u1);
                actualV.actualizacion(nu1val, comboBox1.SelectedIndex, "AInMoPar0",
u2);
                actualV.actualizacion(nu1val, comboBox1.SelectedIndex, "SeNa", u3);
                actualV.actualizacion(nu1val, comboBox1.SelectedIndex, "LPCnt", u4);
                actualV.actualizacion(nu1val, comboBox1.SelectedIndex, "X0", x0);
                actualV.actualizacion(nu1val, comboBox1.SelectedIndex, "Y0", y0);
                actualV.actualizacion(nu1val, comboBox1.SelectedIndex, "X1", x1);
                actualV.actualizacion(nu1val, comboBox1.SelectedIndex, "Y1", y1);
            }
            else
            {
                if (comboBox2.SelectedIndex == 2)
                {
                    uni = "_mA";
                    string u1 = "1";
                    string u2 = "0";
                    string u3 = "Current";
                    string u4 = "2";
                    string x0 = "4E-3";
                    string y0 = "4E-3";
                    string x1 = "2E-2";
                    string y1 = "2E-2";
                    guardarnombres actualV = new guardarnombres();
```

```
                        actualV.actualizacion(nu1val, comboBox1.SelectedIndex, "AInMo",
u1);
                        actualV.actualizacion(nu1val, comboBox1.SelectedIndex,
"AInMoPar0", u2);
                        actualV.actualizacion(nu1val, comboBox1.SelectedIndex, "SeNa",
u3);
                        actualV.actualizacion(nu1val, comboBox1.SelectedIndex, "LPCnt",
u4);
                        actualV.actualizacion(nu1val, comboBox1.SelectedIndex, "X0",
x0);
                        actualV.actualizacion(nu1val, comboBox1.SelectedIndex, "Y0",
y0);
                        actualV.actualizacion(nu1val, comboBox1.SelectedIndex, "X1",
x1);
                        actualV.actualizacion(nu1val, comboBox1.SelectedIndex, "Y1",
y1);
                    }
                    else
                    {
                        uni = "";
                    }
                }
            }
            guardarnombres actual3 = new guardarnombres();
            actual3.actualizacion(nu1val, comboBox1.SelectedIndex, "DaUn", uni);

            string nombre = "@0_" + nu1val + "_c.gcf";
            FTPFactory ff = new FTPFactory();
            ff.setDebug(true);
            ff.setRemoteHost("192.168.2.18");
            ff.setRemoteUser("instrumentation4");
            ff.setRemotePass("gantner");
            ff.login();  //nos conectamos y subimos el archivo
            ff.download("#event.sta");
            ff.upload(nombre);
            ff.upload("#command.sta");
        }

        private void button4_Click(object sender, EventArgs e)
        {
            numericUpDown1.Value = 1;
            guardarnombres desc = new guardarnombres();
            desc.descargar();
            //hacemos un archivo con info sobre las entradas (maximos y mínimos, nombre
y módulo al que pertenecen)
            int contador = 3; //lo ponemos en cuatro para que tenga en cuenta los
contadores que salen luego en la captura
            int k = 0;
            using (XmlWriter writer = XmlWriter.Create("entconfig.xml"))
            {
                writer.WriteStartElement("Configuración");
                writer.WriteElementString("fecha", DateTime.Now.ToString()); //fecha
última actualización
                while (comboBox1.Items.Count >= 1)
                {
                    for (k = 0; k < comboBox1.Items.Count; k++)
                    {
                        comboBox1.SelectedIndex = k;
                        writer.WriteName("Entrada" + contador);
                        writer.WriteElementString("e" + contador + "_Modulo",
numericUpDown1.Value.ToString());
                        writer.WriteElementString("e" + contador + "_Name",
textBox2.Text);
                        writer.WriteElementString("e" + contador + "_Max",
textBox4.Text);
                        writer.WriteElementString("e" + contador + "_Min",
textBox5.Text);
                        contador++;
                    }
                    numericUpDown1.Value++;
                }
                writer.WriteEndElement();
            }
```

```
        }

        private void textBox3_TextChanged(object sender, EventArgs e)
        {
            if (textBox3.Text == "")
            {
                pictureBox1.Hide();
            }
            if (textBox3.Text == "N/A")
            {
                pictureBox1.Hide();
            }
            else
            {
                if (textBox3.Text == "E.Reader")
                {
                    pictureBox1.Load("C:/Documents and Settings/Asier/Mis
documentos/Visual Studio 2008/Projects/menutexto/ereader.bmp");
                    pictureBox1.Show();
                }
                else
                {
                    if (textBox3.Text == "E.Bloxx-Digital")
                    {
                        pictureBox1.Load("C:/Documents and Settings/Asier/Mis
documentos/Visual Studio 2008/Projects/menutexto/ebloxxdig.bmp");
                        pictureBox1.Show();
                    }
                    else
                    {
                        pictureBox1.Load("C:/Documents and Settings/Asier/Mis
documentos/Visual Studio 2008/Projects/menutexto/eblox.bmp");
                        pictureBox1.Show();
                    }
                }
            }
        }
    }
}
```

4. Pantalla captura

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Xml;
using System.Net;

namespace WindowsFormsApplication1
{
    public partial class Form4 : Form
    {
        FTPFactory ff = new FTPFactory();

        public Form4()
        {
            InitializeComponent();
        }
        string textovalores = "";
        int ind = 0;
        int i = 0;
        int contcor = 0;
        int intervalo = 1;
        string b = "";
        string b2 = "";

        private void button3_Click(object sender, EventArgs e)
        {
            Form3 Form3 = new Form3();
            Form3.Visible = true;
            this.Close();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            time.Stop();
            contcor = contcor + 1;
        }

        private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            textBox1.Text = listBox1.SelectedItem.ToString();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            textBox1.Text = "Adquiriendo datos, espere...";
            time.Enabled = true;
            time.Interval = intervalo * 1000; //La primera repetición dura un segundo
            int k = 0;
            i = 0;
            for (k = 0; k <= 1440; k++)
            {
                if (File.Exists(DateTime.Now.Day.ToString() + "-" +
DateTime.Now.Month.ToString() + "-" + DateTime.Now.Year.ToString() + " " + k + ".xml"))
                {
                    i++;
                }
            }
            time.Tick += new EventHandler(time_Tick);
        }
```

```
        private void time_Tick(object sender, EventArgs e)
        {
            if (textBox2.Text == "")
            {
                textBox2.Text = "10";
            }
            int intervalo = Convert.ToInt32(textBox2.Text);
            time.Interval = intervalo * 1000 * 60; //el resto serán en minutos
            this.listBox1.Items.Clear();
            textBox1.Text = "Adquiriendo datos, espere...";
            textBox1.Text = contcor.ToString();
            int k = 0;
            //conexion al módulo
            string remoteUri = "http://192.168.2.18/";
            string fileName = "value.htm", myStringWebResource = null;
            WebClient myWebClient = new WebClient();
            myStringWebResource = remoteUri + fileName;
            myWebClient.DownloadFile(myStringWebResource, fileName);

            string cap = DateTime.Now.Day.ToString()+"-"+
            DateTime.Now.Month.ToString()+"-"+ DateTime.Now.Year.ToString()+" "+ i;
            StringBuilder snom = new StringBuilder(cap);
            snom.Replace("/", "_");
            cap = snom.ToString();
            int j = 0;
            int j2 = 0;
            int j3 = 0;
            string partido="";
            using (StreamReader sr = new StreamReader("value.htm"))
            {
                String line;
                while ((line = sr.ReadLine()) != null)
                {
                    StringBuilder stb = new StringBuilder(line);
                    stb.Replace(" 1)", "/1)");
                    line = stb.ToString();
                    partido = partido + "/" + line;
                }
            }
            j = 1;
            string transf ="";
            using (XmlWriter writer = XmlWriter.Create(cap + ".xml"))
            {
                writer.WriteStartElement("Captura");
                writer.WriteElementString("tiempo", DateTime.Now.ToString()); // la hora
                string[] split2 = partido.Split(new Char[] { '/' });
                foreach (string split3 in split2)
                {
                    StringBuilder sb = new StringBuilder(split3);
                    sb.Replace(" ", "");
                    sb.Replace("    ", "");
                    sb.Replace("", "º");
                    string split5 = sb.ToString();
                    string[] split4 = split5.Split(new Char[] { ' ', '\r', '\n', '\t'
});
                    if (split4[0] == "1)")
                    {
                        writer.WriteElementString("entrada0", split4[2]); //en split4[1]
estan los nombres de las entradas, en split4[2] los datos
                    }
                    if (split4[0] == j + ")")
                    {
                        j3 = j - 1;
                        writer.WriteElementString("entrada" + j3, split4[2]);
                        j++;
                    }
                    transf = transf + "\n" + split5;
                }
                writer.WriteEndElement();
                writer.Flush();
            }
```

```
            //NOMBRES

            int j4 = 2;
            int j5 = 0;
            using (XmlWriter writer2 = XmlWriter.Create("Noment.xml"))
            {
                writer2.WriteStartElement("Captura");
                writer2.WriteElementString("tiempo", DateTime.Now.ToString());
                string[] transfer = transf.Split(new char[] { '\n', '\r' });
                foreach (string splits in transfer)
                {
                    string[] split6 = splits.Split(new Char[] { ' ', '\n', '\t', '\r'
});
                    if (split6[0] == "1)")
                    {
                        writer2.WriteElementString("entrada0", split6[1]);
                    }
                    if (split6[0] == j4 + ")")
                    {
                        j5 = j4 - 1;
                        writer2.WriteElementString("entrada" + j5, split6[1]);
                        j4++;
                    }
                }
                writer2.WriteEndElement();
                writer2.Flush();
            }
            //REPRESENTAR
            listBox1.Items.Clear();
            for (k = 4; k <= 32; k++)
            {
                XmlDocument xmldoc3 = new XmlDocument();
                xmldoc3.Load("entconfig.xml");
                XmlNodeList config = xmldoc3.GetElementsByTagName("e" + k + "_Modulo");
                string b3 = config[0].InnerText.ToString();
                if (b3 == numericUpDown1.Value.ToString())
                {
                    XmlDocument xmldoc = new XmlDocument();
                    xmldoc.Load("Noment.xml");
                    XmlNodeList nombreentrada = xmldoc.GetElementsByTagName("entrada" +
k);
                    b = nombreentrada[0].InnerText;
                    XmlDocument xmldoc2 = new XmlDocument();
                    xmldoc2.Load(cap + ".xml");
                    XmlNodeList vals = xmldoc2.GetElementsByTagName("entrada" + k);
                    b2 = vals[0].InnerText;
                    string linun = b + ":    " + b2;
                    this.listBox1.Items.Add(linun);
                }
            }
            //si el valor de "modulo" está en 0, mostramos todas las entradas
            if (numericUpDown1.Value == 0)
            {
                for (k = 0; k <= 32; k++)
                {
                    XmlDocument xmldoc = new XmlDocument();
                    xmldoc.Load("Noment.xml");
                    XmlNodeList nombreentrada = xmldoc.GetElementsByTagName("entrada" +
k);
                    b = nombreentrada[0].InnerText;
                    XmlDocument xmldoc2 = new XmlDocument();
                    xmldoc2.Load(cap + ".xml");
                    XmlNodeList vals = xmldoc2.GetElementsByTagName("entrada" + k);
                    b2 = vals[0].InnerText;
                    string linun = b + ":    " + b2;
                    this.listBox1.Items.Add(linun);
                }
            }
            i++;
        }
    }
}
```

5. Pantalla gráficos

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using ZedGraph;
using System.Xml;
using System.IO;

namespace WindowsFormsApplication1
{
    public partial class Form5 : Form
    {
        int i = 0;
        string a = "";
        string t = "";
        DateTime ejex = DateTime.Now;
        double ex = 0;
        double adou = 0;
        string b = "";
        string b1 = "";
        string fechainicio = "";
        string fechafinal = "";
        string fechaint = "";
        string fechainiciomod = "";
        int numcapturas = 0;
        int contdias = 0;
        int comparador = 0;
        DateTime fechaini;
        DateTime fechafin;
        double y = 0;
        double y1 = 0;
        double contador = 0;

        private void CreateGraph(ZedGraphControl zgc,string nombreentrada,int
indiceentrada,int numeromuestras)
        {
            TimeSpan iteraciones = fechafin.Subtract(fechaini);
            string str = iteraciones.ToString();
            string [] str2 = str.Split(new char [] {'.',':','-'});
            if (str2[0] == "")
            {
                str2[0] = "0";
            }
            contdias = Convert.ToInt32(str2[0]);
            //textBox1.Text = contdias.ToString();
            GraphPane myPane = zgc.GraphPane;
            myPane.CurveList.Clear();
            // Configuramos el eje x con formato de fecha
            myPane.XAxis.Type = AxisType.Date;
            // Set the titles and axis labels
            myPane.Title.Text = nombreentrada;
            myPane.XAxis.Title.Text = "Tiempo";
            myPane.YAxis.Title.Text = "Unidades";
            myPane.Legend.IsVisible=false;
            // Make up some data points from the Sine function
            PointPairList list = new PointPairList();
            PointPairList listmax = new PointPairList();
            PointPairList listmin = new PointPairList();
                comparador = numeromuestras;
                //textBox2.Text = comparador.ToString();
                while (comparador != -1)
                {
                    int x = 0;
```

```
                    fechainicio = fechaini.Day.ToString() + "-" +
fechaini.Month.ToString() + "-" + fechaini.Year.ToString();
                    while (File.Exists(fechainicio + " " + x + ".xml"))
                    {
                        XmlDocument xmldoc = new XmlDocument();
                        xmldoc.Load(fechainicio + " " + x + ".xml");
                        XmlNodeList Valores = xmldoc.GetElementsByTagName("entrada" +
indiceentrada);
                        a = Valores[0].InnerText;
                        XmlDocument xmldoc2 = new XmlDocument();
                        xmldoc2.Load(fechainicio + " " + x + ".xml");
                        XmlNodeList Tiempo = xmldoc2.GetElementsByTagName("tiempo");
                        t = Tiempo[0].InnerText;
                        if (a == "---")
                        {
                            a = "0";
                            myPane.YAxis.Title.Text = "N/A";
                        }
                        else
                        {
                            if (a == "false")
                            {
                                a = "0";
                                myPane.YAxis.Title.Text = "Bool";
                            }
                        }
                        if (a.Contains('V'))
                        {
                            myPane.YAxis.Title.Text = "Voltios (V)";
                        }
                        if (a.Contains('C'))
                        {
                            myPane.YAxis.Title.Text = "Grados (ºC)";
                        }
                        if (a.Contains('A'))
                        {
                            myPane.YAxis.Title.Text = "Amperios (A)";
                        }
                        a = a.Replace('.', ','); //cambiamos el punto por una coma
                        a = a.Replace("V", "");
                        a = a.Replace("ºC", "");
                        a = a.Replace("ºC", "");
                        a = a.Replace("_mA", "");
                        a = a.Replace("mA", "");
                        adou = Convert.ToDouble(a);
                        ejex = Convert.ToDateTime(t);
                        ex = ejex.ToOADate();
                        list.Add(ex, adou);
                    //Generamos una curva
                    LineItem myCurve = myPane.AddCurve("My Curve", list, Color.Black,
SymbolType.Circle);
                    //Llenamos la parte inferior de la curva con un gradiente gris
                    myCurve.Line.Fill = new Fill(Color.Gainsboro, Color.Transparent,
90F);
                    myCurve.Line.Width = 1;
                    myCurve.Symbol.Fill = new Fill(Color.White);
                    //Llenamos la gráfica con un gradiente de color de fondo
                    myPane.Chart.Fill = new Fill(Color.White,
Color.LightGoldenrodYellow, 45F);
                    myPane.Fill = new Fill(Color.White, Color.FromArgb(220, 220, 255),
45F);
                    //Calculamos los ejes
                    zgc.AxisChange();
                        x++;
                        contador++;

                        //Máximo y mínimo

                        if (textBox1.Text == "N/A")
                        {
                            textBox1.Text = "0";
                            textBox2.Text = "0";
                        }
```

```
                        listmax.Add(ex, Convert.ToDouble(textBox1.Text));
                        LineItem myCurvemax = myPane.AddCurve("", listmax, Color.Red,
SymbolType.None);
                        myCurvemax.Line.Width = 5;
                        listmin.Add(ex, Convert.ToDouble(textBox2.Text));
                        LineItem myCurvemin = myPane.AddCurve("", listmin, Color.Orange,
SymbolType.None);
                        myCurvemin.Line.Width = 5;

                    }
                    fechaini = fechaini.AddDays(1);
                    if (contdias == 0)
                    {
                        comparador = -1;
                    }
                    contdias--;
                }
        }

        public Form5()
        {
            InitializeComponent();
        }

        private void button3_Click(object sender, EventArgs e)
        {
            Form3 Form3 = new Form3();
            Form3.Visible = true;
            this.Close();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Form6 Form6 = new Form6();
            Form6.Visible = true;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            fechaini = dateTimePicker1.Value;
            fechafin = dateTimePicker2.Value;
            this.comboBox1.Items.Clear(); //evitamos que se repitan las entradas
            for (i=0;i<=32;i++)
            {
                XmlDocument xmldoc = new XmlDocument();
                xmldoc.Load("Noment.xml");
                XmlNodeList Nombres = xmldoc.GetElementsByTagName("entrada"+i);
                b = Nombres[0].InnerText;
                this.comboBox1.Items.Add(b);
            }
        }

        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            fechaini = dateTimePicker1.Value;
            fechafin = dateTimePicker2.Value;
            numcapturas = fechafin.CompareTo(fechaini);
            textBox1.Text="0";
            textBox2.Text="0";
            if (comboBox1.SelectedIndex >= 3)
            {
                XmlDocument xmldoc3max = new XmlDocument();
                xmldoc3max.Load("entconfig.xml");
                XmlNodeList maximo = xmldoc3max.GetElementsByTagName("e" +
comboBox1.SelectedIndex + "_Max");
                string maxi = maximo[0].InnerText;
                textBox1.Text = maxi;
                XmlDocument xmldoc3min = new XmlDocument();
                xmldoc3min.Load("entconfig.xml");
                XmlNodeList minimo = xmldoc3min.GetElementsByTagName("e" +
comboBox1.SelectedIndex + "_Min");
                string mini = minimo[0].InnerText;
```

```
                textBox2.Text = mini;
            }
            contador = 0;
            CreateGraph(zedGraphControl1, comboBox1.Text,
comboBox1.SelectedIndex,numcapturas);
        }
    }
}
```

6. Biblioteca de clases y punto de entrada

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using System.IO;
using System.Text;
using System.Net;
using System.Net.Sockets;


namespace WindowsFormsApplication1
{
    public class guardarnombres
    {
        string vacio = "";

        // Método nombres
        public String nombres(int nummodulo, string palabraclave)
        {
            string[] NOMBRE;
            NOMBRE = new string[1000];
            string palabra = palabraclave;
            int i = 1;
            i = nummodulo;
            int k = 0;
            int cont = 0;
            String[] Nom = new String[500];
            if (File.Exists("@0_" + i + "_c.gcf"))
            {
                try
                {
                    //leemos el archivo linea por linea
                    using (StreamReader sr = new StreamReader("@0_" + i + "_c.gcf"))
                    {
                        String line;
                        //guardamos cada linea en line
                        while ((line = sr.ReadLine()) != null)
                        {
                            cont = 0;
                            //dividimos la linea en palabras
                            string[] split = line.Split(new Char[] { ')', ' ', ',', '.',
':', '=', '<', '>' });
                            //si la primera palabra es "Na" guardamos valor en “Nom”
                            if (split[0] == palabra)
                            {
                                Nom[cont] = split[1];
                                //vamos almacenando los valores separado por una barra
                                for (k = 0; k <= cont; k++)
                                {
                                    NOMBRE[i] = NOMBRE[i] + Nom[k] + "/";
                                }
                            }
                            cont = cont + 1;
                        }
                    }
                }

                catch (Exception ex)
                {
                    //-------
                }
            }
            else
            {
                //si el archivo no existe ponemos la salida a cero
                NOMBRE[i] = "";
            }
```

```
            //devolvemos NOMBRE[i] al programa
            return NOMBRE[i];
        }

        public string reconocimiento(int nummodulo, string info)
        {
            string[] RECON;
            RECON = new string[1000];
            string palabraclave = "";
            palabraclave = info;
            int i = 1;
            i = nummodulo;
            int k = 0;
            int cont = 0;
            String[] Rec = new String[500];
            if (File.Exists("@0_" + i + "_c.gcf"))
            {
            try
            {
                using (StreamReader sr = new StreamReader("@0_" + i + "_c.gcf"))
                {
                    String line;
                    while ((line = sr.ReadLine()) != null)
                    {
                        cont = 0;
                        string[] split = line.Split(new Char[] { ')', ' ', ',', '.',
':', '=', '<', '>','[',']','/' });
                        if (split[0] == palabraclave)
                        {
                            Rec[cont] = split[1];
                            for (k = 0; k <= cont; k++)
                            {
                                RECON[i] = RECON[i] + Rec[k] +"/";
                            }
                        }
                        cont = cont + 1;
                    }
                }
            }
            catch (Exception ex)
            {
            }
            return RECON[i];
            }
            return vacio;
        }

        public string valores(string info)
        {
            string[] RECON;
            RECON = new string[1000];
            string palabraclave = "";
            palabraclave = info;
            int k = 0;
            int cont = 0;
            String[] Rec = new String[500];
            string line2 = "";
            String[] line3 = new String[500];
            string line4 = "";
            string line5 = "";
            try
            {
                using (StreamReader sr = new StreamReader("$onlasc.dat"))
                {
                    String line;
                    while ((line = sr.ReadLine()) != null)
                    {
                        line2 = line2 + line + "/";
                        cont = 0;
                        string[] split2 = line2.Split(new Char[] { '/'});
                        split2.CopyTo(line3, 0);
                    }
                    for (k = 0; k <= line3.Length; k++)
```

```
                    {
                        line4 = line4 + line3[k];
                    }
                }
            }
            catch (Exception ex)
            {
            }

            return line2;
        }

        public void descargar()
        {
            FTPFactory ff = new FTPFactory();
            ff.setDebug(true);
            ff.setRemoteHost("192.168.2.18");
            ff.setRemoteUser("instrumentation4");
            ff.setRemotePass("gantner");
            ff.login();
            int i = 0;
            string archivoac = "";
            string[] fileNames = ff.getFileList("*.*");
            string[] lista = ff.getFileList("*.*");
            if (File.Exists("listado.txt"))
            {
                File.Delete("listado.txt");
            }
            //creamos el archivo si este no existe y escribimos en él
            using (StreamWriter sw = File.CreateText("listado.txt"))
            {
                for (i = 0; i <= lista.Length - 1; i++)
                {
                    sw.WriteLine(fileNames[i]);
                }
            }

            try
            {
                using (StreamReader sr = new StreamReader("listado.txt"))
                {
                    String line;
                    while ((line = sr.ReadLine()) != null)
                    {
                        string [] split = line.Split(new Char[] {'\n', '/','\r',' ' });
                        for (i = 0; i <= lista.Length - 1; i++)
                        {
                            archivoac = "@0_"+i+"_c.gcf";
                            if (split[0] == archivoac)
                            {
                                ff.download(split[0]);
                            }
                        }
                    }
                }
            }
            catch (Exception ex)
            {
            }
        }

        public void actualizacion(int indicemodulo, int indiceentrada, string
palabraclave, string nuevovalor)
        {
            int cont = 0;
            string a = "";
            string nombre = "@0_"+indicemodulo+"_c.gcf";
```

```
            using (StreamReader sr = new StreamReader(nombre)) //hay que darle el nombre
del archivo de origen
            {

                String line;
                while ((line = sr.ReadLine()) != null)
                {
                    //dividimos la cadena lineen palabras
                    string[] split = line.Split(new Char[] { ')', ' ', ',', '.', ':',
'=', '<', '>' });
                    if (split[0] == palabraclave)   //Na es la palabra clave
                    {
                        if (cont == indiceentrada) //hay que darle el indice de la
entrada
                        {
                            line = palabraclave+"="+nuevovalor; // "Na=Entrada_" + cont
es el nuevo valor
                        }
                        cont = cont + 1;
                    }
                    if (a == "")
                    {
                        a = a+line;
                    }
                    else
                    {
                        a = a + "\r" + line;
                    }
                    using (StreamWriter sw = new StreamWriter("texto2")) //el nombre del
archivo destino
                    {
                        sw.WriteLine(a);
                        sw.Close();
                    }
                }
            }
            if (File.Exists(nombre))
            {
                File.Delete(nombre);
            }
            File.Copy("texto2", nombre); //reescribimos el original
            //Creamos el archivo command que acompañará al archivo de configuración
            using (StreamWriter com = new StreamWriter("#command.sta")) //el nombre del
archivo destino
            {
                com.WriteLine("TRANSFER_MASTER2SLAVE       0       "+indicemodulo);
            }
        }
    }
```

```
        public class FTPFactory
        {

            private string
        remoteHost, remotePath, remoteUser, remotePass, mes;
            private int remotePort, bytes;
            private Socket clientSocket;

            private int retValue;
            private Boolean debug;
            private Boolean logined;
            private string reply;

            private static int BLOCK_SIZE = 512;

            Byte[] buffer = new Byte[BLOCK_SIZE];
            Encoding ASCII = Encoding.ASCII;
```

```
public FTPFactory()
{

    remoteHost = "localhost";
    remotePath = ".";
    remoteUser = "anonymous";
    remotePass = "password";
    remotePort = 21;
    debug = false;
    logined = false;

}

//
// Set the name of the FTP server to connect to.
//
// Server name
public void setRemoteHost(string remoteHost)
{
    this.remoteHost = remoteHost;
}

//
// Return the name of the current FTP server.
//
// Server name
public string getRemoteHost()
{
    return remoteHost;
}

//
// Set the port number to use for FTP.
//
// Port number
public void setRemotePort(int remotePort)
{
    this.remotePort = remotePort;
}

//
// Return the current port number.
//
// Current port number
public int getRemotePort()
{
    return remotePort;
}

//
// Set the remote directory path.
//
// The remote directory path
public void setRemotePath(string remotePath)
{
    this.remotePath = remotePath;
}

//
// Return the current remote directory path.
//
// The current remote directory path.
public string getRemotePath()
{
    return remotePath;
}

//
// Set the user name to use for logging into the remote server.
```

```
// Username
public void setRemoteUser(string remoteUser)
{
    this.remoteUser = remoteUser;
}

//
// Set the password to user for logging into the remote server.
//
// Password
public void setRemotePass(string remotePass)
{
    this.remotePass = remotePass;
}

//
// Return a string array containing the remote directory's file list.
//
//
//
public string[] getFileList(string mask)
{

    if (!logined)
    {
        login();
    }

    Socket cSocket = createDataSocket();

    sendCommand("NLST " + mask);

    if (!(retValue == 150 || retValue == 125))
    {
        throw new IOException(reply.Substring(4));
    }

    mes = "";

    while (true)
    {

        int bytes = cSocket.Receive(buffer, buffer.Length, 0);
        mes += ASCII.GetString(buffer, 0, bytes);

        if (bytes < buffer.Length)
        {
            break;
        }
    }

    char[] seperator = { '\n' };
    string[] mess = mes.Split(seperator);

    cSocket.Close();

    readReply();

    if (retValue != 226)
    {
        throw new IOException(reply.Substring(4));
    }
    return mess;

}
```

```
        //
        // Return the size of a file.
        //
        //
        //
        public long getFileSize(string fileName)
        {

            if (!logined)
            {
                login();
            }

            sendCommand("SIZE " + fileName);
            long size = 0;

            if (retValue == 213)
            {
                size = Int64.Parse(reply.Substring(4));
            }
            else
            {
                throw new IOException(reply.Substring(4));
            }

            return size;

        }

        //
        // Login to the remote server.
        //
        public void login()
        {

            clientSocket = new
    Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
            IPEndPoint ep = new
    IPEndPoint(Dns.Resolve(remoteHost).AddressList[0], remotePort);

            try
            {
                clientSocket.Connect(ep);
            }
            catch (Exception)
            {
                throw new IOException("Couldn't connect to remote server");
            }

            readReply();
            if (retValue != 220)
            {
                close();
                throw new IOException(reply.Substring(4));
            }
            if (debug)
                Console.WriteLine("USER " + remoteUser);

            sendCommand("USER " + remoteUser);

            if (!(retValue == 331 || retValue == 230))
            {
                cleanup();
                throw new IOException(reply.Substring(4));
            }

            if (retValue != 230)
            {
                if (debug)
                    Console.WriteLine("PASS xxx");

                sendCommand("PASS " + remotePass);
                if (!(retValue == 230 || retValue == 202))
```

```
            {
                cleanup();
                throw new IOException(reply.Substring(4));
            }
        }

        logined = true;
        Console.WriteLine("Connected to " + remoteHost);

        chdir(remotePath);

    }

    //
    // If the value of mode is true, set binary mode for downloads.
    // Else, set Ascii mode.
    //
    //
    public void setBinaryMode(Boolean mode)
    {

        if (mode)
        {
            sendCommand("TYPE I");
        }
        else
        {
            sendCommand("TYPE A");
        }
        if (retValue != 200)
        {
            throw new IOException(reply.Substring(4));
        }
    }

    //
    // Download a file to the Assembly's local directory,
    // keeping the same file name.
    //
    //
    public void download(string remFileName)
    {
        download(remFileName, "", false);
    }

    //
    // Download a remote file to the Assembly's local directory,
    // keeping the same file name, and set the resume flag.
    //
    //
    //
    public void download(string remFileName, Boolean resume)
    {
        download(remFileName, "", resume);
    }

    //
    // Download a remote file to a local file name which can include
    // a path. The local file name will be created or overwritten,
    // but the path must exist.
    //
    //
    //
    public void download(string remFileName, string locFileName)
    {
        download(remFileName, locFileName, false);
    }

    //
    // Download a remote file to a local file name which can include
    // a path, and set the resume flag. The local file name will be
    // created or overwritten, but the path must exist.
    //
```

```
            public void download(string remFileName, string
        locFileName, Boolean resume)
            {
                if (!logined)
                {
                    login();
                }

                setBinaryMode(true);

                Console.WriteLine("Downloading file " + remFileName + " from " +
remoteHost + "/" + remotePath);

                if (locFileName.Equals(""))
                {
                    locFileName = remFileName;
                }

                if (!File.Exists(locFileName))
                {
                    Stream st = File.Create(locFileName);
                    st.Close();
                }

                FileStream output = new
          FileStream(locFileName, FileMode.Open);

                Socket cSocket = createDataSocket();

                long offset = 0;

                if (resume)
                {

                    offset = output.Length;

                    if (offset > 0)
                    {
                        sendCommand("REST " + offset);
                        if (retValue != 350)
                        {
                            //throw new IOException(reply.Substring(4));
                            //Some servers may not support resuming.
                            offset = 0;
                        }
                    }

                    if (offset > 0)
                    {
                        if (debug)
                        {
                            Console.WriteLine("seeking to " + offset);
                        }
                        long npos = output.Seek(offset, SeekOrigin.Begin);
                        Console.WriteLine("new pos=" + npos);
                    }
                }

                sendCommand("RETR " + remFileName);

                if (!(retValue == 150 || retValue == 125))
                {
                    throw new IOException(reply.Substring(4));
                }

                while (true)
                {

                    bytes = cSocket.Receive(buffer, buffer.Length, 0);
                    output.Write(buffer, 0, bytes);

                    if (bytes <= 0)
                    {
```

```
                break;
            }
        }

        output.Close();
        if (cSocket.Connected)
        {
            cSocket.Close();
        }

        Console.WriteLine("");

        readReply();

        if (!(retValue == 226 || retValue == 250))
        {
            throw new IOException(reply.Substring(4));
        }

    }

    //
    // Upload a file.
    //
    //
    public void upload(string fileName)
    {
        upload(fileName, false);
    }

    //
    // Upload a file and set the resume flag.
    //
    //
    //
    public void upload(string fileName, Boolean resume)
    {

        if (!logined)
        {
            login();
        }

        Socket cSocket = createDataSocket();
        long offset = 0;

        if (resume)
        {

            try
            {

                setBinaryMode(true);
                offset = getFileSize(fileName);

            }
            catch (Exception)
            {
                offset = 0;
            }
        }

        if (offset > 0)
        {
            sendCommand("REST " + offset);
            if (retValue != 350)
            {
                //throw new IOException(reply.Substring(4));
                //Remote server may not support resuming.
                offset = 0;
            }
        }
```

```
            sendCommand("STOR " + Path.GetFileName(fileName));

            if (!(retValue == 125 || retValue == 150))
            {
                throw new IOException(reply.Substring(4));
            }

            // open input stream to read source file
            FileStream input = new
        FileStream(fileName, FileMode.Open);

            if (offset != 0)
            {

                if (debug)
                {
                    Console.WriteLine("seeking to " + offset);
                }
                input.Seek(offset, SeekOrigin.Begin);
            }

            Console.WriteLine("Uploading file " + fileName + " to " + remotePath);

            while ((bytes = input.Read(buffer, 0, buffer.Length)) > 0)
            {

                cSocket.Send(buffer, bytes, 0);

            }
            input.Close();

            Console.WriteLine("");

            if (cSocket.Connected)
            {
                cSocket.Close();
            }

            readReply();
            if (!(retValue == 226 || retValue == 250))
            {
                throw new IOException(reply.Substring(4));
            }
        }

        //
        // Delete a file from the remote FTP server.
        //
        //
        public void deleteRemoteFile(string fileName)
        {

            if (!logined)
            {
                login();
            }

            sendCommand("DELE " + fileName);

            if (retValue != 250)
            {
                throw new IOException(reply.Substring(4));
            }

        }
```

```
        //
        // Rename a file on the remote FTP server.
        //
        //
        //
        public void renameRemoteFile(string oldFileName, string
    newFileName)
        {

            if (!logined)
            {
                login();
            }

            sendCommand("RNFR " + oldFileName);

            if (retValue != 350)
            {
                throw new IOException(reply.Substring(4));
            }

            //  known problem
            //  rnto will not take care of existing file.
            //  i.e. It will overwrite if newFileName exist
            sendCommand("RNTO " + newFileName);
            if (retValue != 250)
            {
                throw new IOException(reply.Substring(4));
            }

        }

        //
        // Create a directory on the remote FTP server.
        //
        //
        public void mkdir(string dirName)
        {

            if (!logined)
            {
                login();
            }

            sendCommand("MKD " + dirName);

            if (retValue != 250)
            {
                throw new IOException(reply.Substring(4));
            }

        }

        //
        // Delete a directory on the remote FTP server.
        //
        //
        public void rmdir(string dirName)
        {

            if (!logined)
            {
                login();
            }

            sendCommand("RMD " + dirName);

            if (retValue != 250)
            {
                throw new IOException(reply.Substring(4));
            }

        }
```

```
//
// Change the current working directory on the remote FTP server.
//
//
public void chdir(string dirName)
{

    if (dirName.Equals("."))
    {
        return;
    }

    if (!logined)
    {
        login();
    }

    sendCommand("CWD " + dirName);

    if (retValue != 250)
    {
        throw new IOException(reply.Substring(4));
    }

    this.remotePath = dirName;

    Console.WriteLine("Current directory is " + remotePath);

}

//
// Close the FTP connection.
//
public void close()
{

    if (clientSocket != null)
    {
        sendCommand("QUIT");
    }

    cleanup();
    Console.WriteLine("Closing...");
}

//
// Set debug mode.
//
//
public void setDebug(Boolean debug)
{
    this.debug = debug;
}

private void readReply()
{
    mes = "";
    reply = readLine();
    retValue = Int32.Parse(reply.Substring(0, 3));
}

private void cleanup()
{
    if (clientSocket != null)
    {
        clientSocket.Close();
        clientSocket = null;
    }
    logined = false;
}

private string readLine()
```

```
        {

            while (true)
            {
                bytes = clientSocket.Receive(buffer, buffer.Length, 0);
                mes += ASCII.GetString(buffer, 0, bytes);
                if (bytes < buffer.Length)
                {
                    break;
                }
            }

            char[] seperator = { '\n' };
            string[] mess = mes.Split(seperator);

            if (mes.Length > 2)
            {
                mes = mess[mess.Length - 2];
            }
            else
            {
                mes = mess[0];
            }

            if (!mes.Substring(3, 1).Equals(" "))
            {
                return readLine();
            }

            if (debug)
            {
                for (int k = 0; k < mess.Length - 1; k++)
                {
                    Console.WriteLine(mess[k]);
                }
            }
            return mes;
        }

        private void sendCommand(String command)
        {

            Byte[] cmdBytes =
    Encoding.ASCII.GetBytes((command + "\r\n").ToCharArray());
            clientSocket.Send(cmdBytes, cmdBytes.Length, 0);
            readReply();
        }

        private Socket createDataSocket()
        {
            sendCommand("PASV");

            if (retValue != 227)
            {
                throw new IOException(reply.Substring(4));
            }

            int index1 = reply.IndexOf('(');
            int index2 = reply.IndexOf(')');
            string ipData =
    reply.Substring(index1 + 1, index2 - index1 - 1);
            int[] parts = new int[6];

            int len = ipData.Length;
            int partCount = 0;
            string buf = "";

            for (int i = 0; i < len && partCount <= 6; i++)
            {

                char ch = Char.Parse(ipData.Substring(i, 1));
                if (Char.IsDigit(ch))
                    buf += ch;
```

```
                else if (ch != ',')
                {
                    throw new IOException("Malformed PASV reply: " +
          reply);
                }

                if (ch == ',' || i + 1 == len)
                {

                    try
                    {
                        parts[partCount++] = Int32.Parse(buf);
                        buf = "";
                    }
                    catch (Exception)
                    {
                        throw new IOException("Malformed PASV reply: " +
            reply);
                    }
                }
            }

            string ipAddress = parts[0] + "." + parts[1] + "." +
              parts[2] + "." + parts[3];

            int port = (parts[4] << 8) + parts[5];

            Socket s = new
     Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
            IPEndPoint ep = new
     IPEndPoint(Dns.Resolve(ipAddress).AddressList[0], port);

            try
            {
                s.Connect(ep);
            }
            catch (Exception)
            {
                throw new IOException("Can't connect to remote server");
            }

            return s;
        }
    }

}
```

```
static class Program
{
    /// <summary>
    /// Punto de entrada principal para la aplicación.
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new WindowsFormsApplication1.Form1());
    }
}
```

Pamplona, Septiembre de 2010