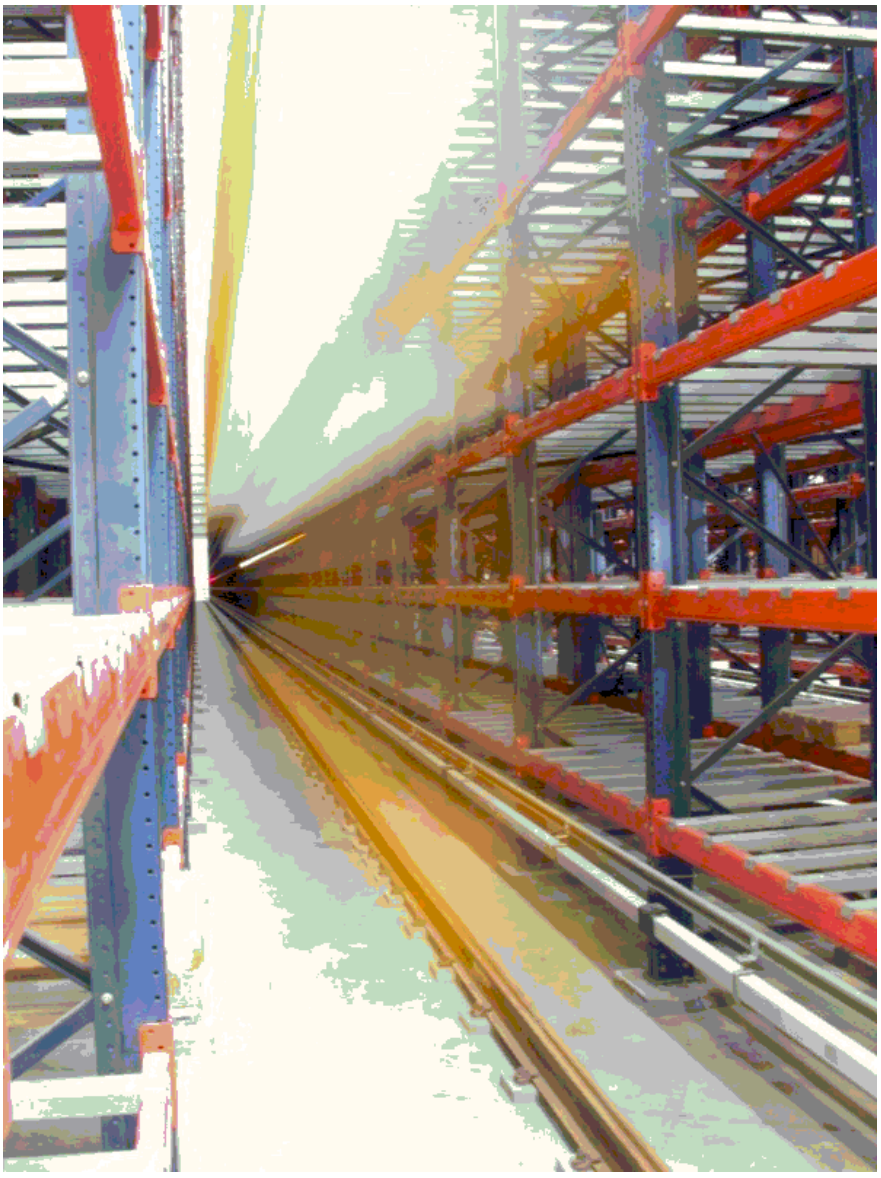
SISTEMA DE CONTROL GALILEO

DOCUMENTACIÓN. MANUAL DE USUARIO



Índice de Revisiones

Fecha	Edición	Autor	Concepto
Agosto 2000	1.0	Jose Luis Santiago	Documento inicial del Sistema de Control Galileo versión 2.0
Diciembre 2001	2.0	Jose Luis Santiago	Se añaden nuevos componentes y la librería de comunicaciones al SCG versión 2.2
Abril 2002	3.0	Jose Luis Santiago	Se actualizan cambios del SCG versión 2.3
Marzo 2004	4.0	Manuel Jesús Fernández	Actualización del documento del SCG versión 3.0
Septiembre 2004	5.0	Manuel Jesús Fernández	Nuevos métodos y funcionalidades del SCG versión 3.0
Mayo 2007	6.0	Manuel Jesús Fernández	Actualización documentación del SCG versión 3.1
Octubre 2007	7.0	Javier Piñera	Nuevos métodos y funcionalidades del SCG versión 3.1
Mayo 2009	8.0	Javier Piñera	Se añaden nuevas funcionalidades del SCG versión 3.1
Enero 2010	8.1	Montse Cañedo	Cambio formato y generación de fichero de ayuda del SCG versión 3.1.

Tabla de Contenido

Introducción

El siguiente documento tiene como objetivo describir el conjunto del ***Sistema de Control Galileo*** explicando detalladamente las capas que lo componen, la

interrelación entre ellas y cómo esta interrelación permite el control de instalaciones complejas.

Previamente a describir el sistema hay que hacer notar que el Sistema de Control Galileo solamente funciona con instalaciones realizadas a través de periferia distribuida, por lo que será conveniente empezar explicando los elementos básicos de los buses de campo y su modo de funcionamiento.

1 BUSES DE CAMPO

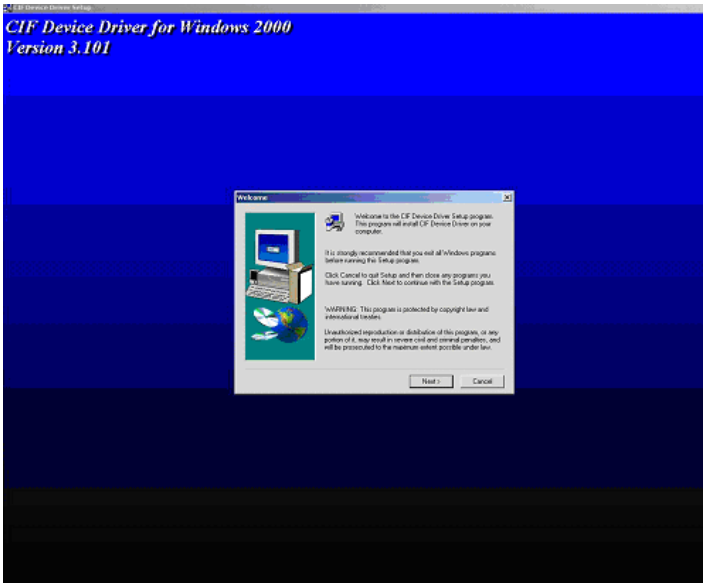
1.1 Instalación de la tarjeta de control Hilscher

La tarjeta de control utilizada habitualmente es de la firma Hilscher (www.hilscher.com) siendo el modelo dependiente del bus de campo utilizado (descrito en los siguientes apartados). Esta tarjeta es PCI y tiene las siguientes características:

- La tarjeta tiene una gestión de comunicación con el sistema host mediante un procesador que se encuentra en la propia tarjeta. El fabricante Hilscher ha diseñado una tarjeta que permite comunicación con diferentes sistemas de bus de campo, soporta los siguientes: **Interbus-S**, **Profibus DP**, **Profibus FMS**, **CanBus**, **CanOpen**, **DeviceNet**, **ASI**, **ControlNet**, **ModConnect**, **Smart Distributed System**. Este procesador se encarga de la gestión del bus estableciendo un protocolo con el sistema Host a través de una DPM (Dual Port Memory).
- Acepta comunicación mediante polling o interrupción.
- Dispone de un sistema de configuración y diagnóstico denominado System Configurator (SyCon).
- Es compatible con varios sistemas operativos.
- Soporta interrupciones compartidas (Shared Interrupt).

Para la instalación de la tarjeta se utilizará el *Setup* del fabricante:

1. En el directorio del driver de Hilscher se debe ejecutar la aplicación *Setup.exe* esto iniciará la instalación.

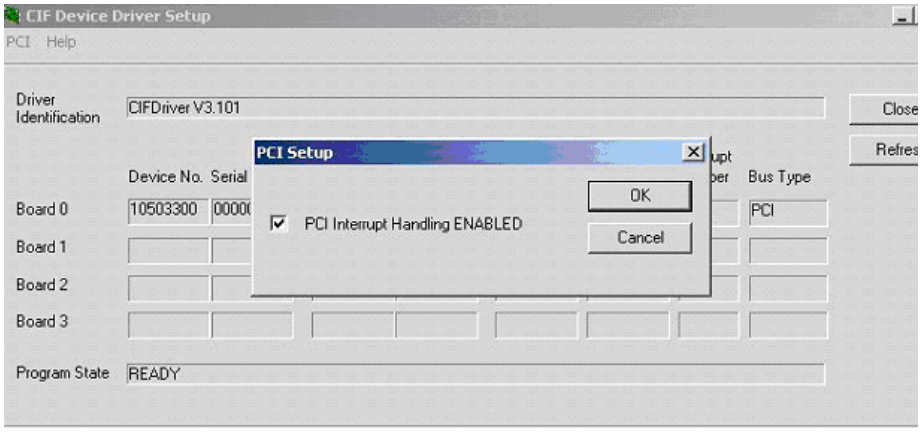


2. Pulsando sobre **Next>** en la pantalla se comprobará en el disco duro si existen versiones viejas de las librerías de enlace dinámico del driver para evitar problemas posteriores.

Si el programa de instalación encuentra versiones viejas de las librerías nos dará la opción de eliminarlas, renombrarlas o conservarlas. *La opción que se debe escoger es eliminarlas.*

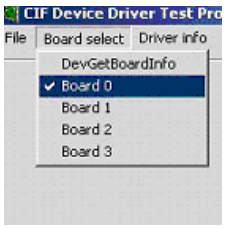
Debemos hacer notar que en versiones previas a la versión 1.7 del Sistema de Control Galileo sólo soportaba una tarjeta de comunicaciones. *Actualmente el sistema admite instalar hasta 4 tarjetas de comunicaciones. Estas tarjetas conformarán una periferia única que se mapea sobre una única imagen de proceso.*

3. Se ejecuta la aplicación *CIF Device Driver Setup* y se selecciona la opción *PCI->Setup,* donde se deshabilita el modo interrupción.

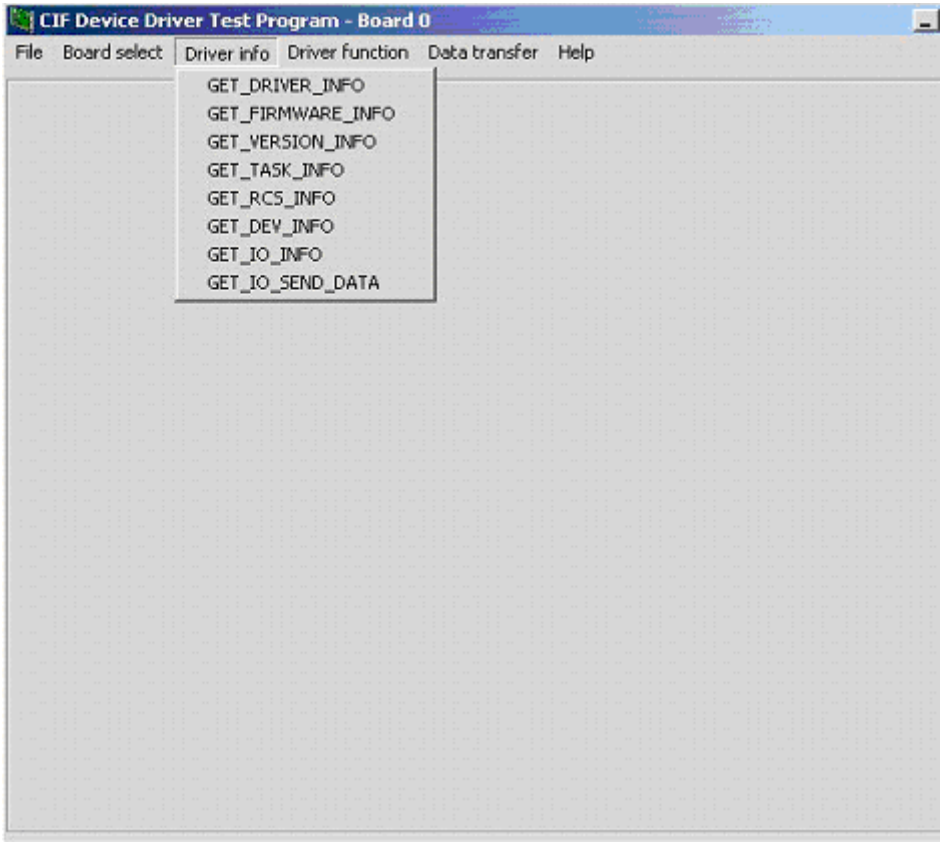


Una vez reseteada la máquina este será el modo fijado. La interrupción seleccionada no puede ser modificada ya que el propio driver decide en cual se va a establecer.

4. Se inicia la aplicación *CIF Device Driver Test* y se selecciona la tarjeta.



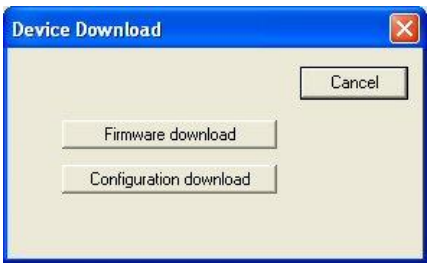
A partir de este momento se pueden utilizar todas las funciones del driver desde la aplicación de test. Lo cual nos puede reportar un nivel de diagnosis bastante elevado si se presentan problemas.



La primera comprobación a realizar consiste en determinar si la versión de firmware instalado en la tarjeta es la última disponible. Para ello ejecutando sobre la opción GET_FIRMWARE_INFO tendremos el resultado de versión y fecha de revisión.

En el caso de tener que efectuar dicha actualización procederíamos de la siguiente manera. Una vez seleccionada la tarjeta como se ha descrito anteriormente, se selecciona la opción:

Driver Function → DevDownload en la cual nos aparecerá el siguiente cuadro de dialogo.



En el seleccionaremos *Firmware* y a continuación seleccionaremos el archivo de firmware que deseamos transferir.

NOTA: *Debe tenerse en cuenta que toda transferencia de Firmware lleva implícita un borrado de la memoria del procesador de la tarjeta, y por lo tanto una pérdida de los archivos de configuración, es decir es necesario volver a transferir la configuración de bus a la tarjeta.*

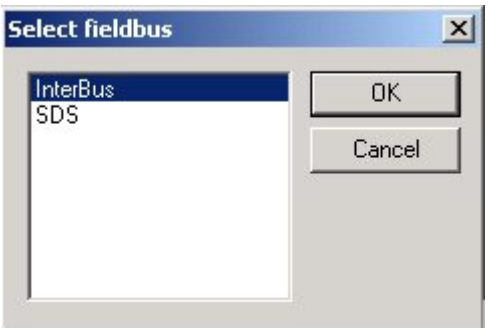
1.1.1 Instalación de la aplicación SyCon (System Configurator)

El siguiente paso es la instalación del Software de configuración de la tarjeta de control. Este Software en el caso de las tarjetas Hilscher se denomina System Configurator y permite las opciones comunes de:

- Detección del bus de campo configurado.
- Visualización de la periferia.
- Diagnosis ante problemas de bus.
- Exportación de la configuración de bus para su utilización directa por el Designer.

La instalación es la normal en todas las aplicaciones de Windows. Incorpora un Setup que instalará el programa. Esta aplicación nos permite realizar las acciones normales para configurar el bus, detectar la configuración, chequear el estado del mismo mediante diagnóstico y comprobar el estado de Entradas / salidas.

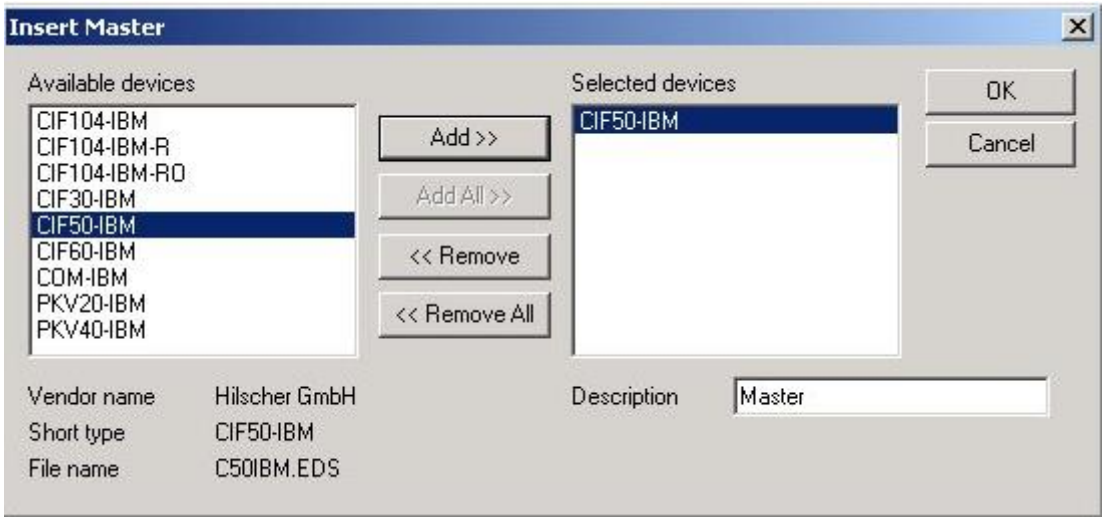
Al abrir un nuevo proyecto en el Sycon se debe seleccionar el bus de campo que se utilizará:



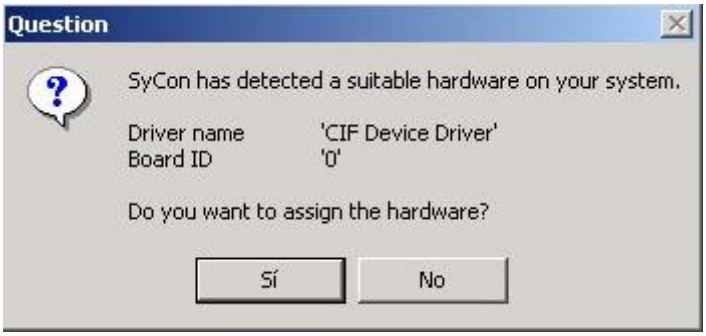
Una vez seleccionado aparece la pantalla de configuración de bus. Esta pantalla aparece vacía y lo primero que debemos hacer es insertar una tarjeta controladora maestra.

En este momento se nos presentará una pantalla en la que debemos elegir el tipo de tarjeta a utilizar.

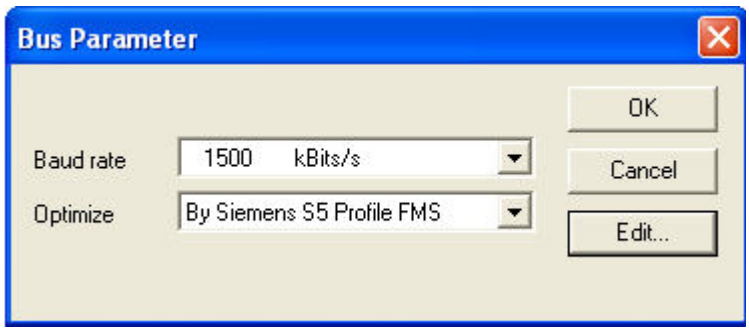
NOTA: *Debe tenerse en cuenta que el tipo de tarjeta seleccionado no debe de ser cambiado. Si se cambia pueden aparecer problemas en función del tipo de configuración de bus que se seleccione.*



Una vez seleccionada la tarjeta, y si coincide que está instalada en el sistema, da la opción de vincularla para que pueda ponerse online de forma directa.



Las operaciones más importantes a realizar sobre la tarjeta aparecen en el menú de *Online***,** donde veremos todas las opciones que se pueden aplicar a la misma. Sin embargo, en el caso de configurar una tarjeta PROFIBUS, hay un paso que debe hacerse obligatoriamente en este momento, y es configurar la velocidad de Bus. Para ello, usamos la opción "*Settings → Bus Parameters*" del menú principal, lo que hace que aparezca el siguiente diálogo:



En el caso de PROFIBUS *debemos elegir en este momento la opción de Siemens S5 profile FMS* si queremos optar a una velocidad de transmisión adecuada (1500 kbits/seg) ya que en cuanto realicemos cualquier otro cambio en la configuración de la tarjeta, perderemos la opción de elegir este perfil, y los otro perfiles no funcionarán adecuadamente y tendremos que configurar cada uno de los parámetros de bus de forma manual.

La primera opción por importancia es *Automatic Network Scan*..., con esta opción la tarjeta realizará una diagnosis del bus reconociendo los elementos que se encuentran conectados al mismo. De esta forma se puede realizar una configuración rápida y sin problemas.

Una vez configurado el bus otras de las opciones importantes para monitorizar son:

- *I/O Monitor*: Mostrará las entradas salidas activas.
- *Start Debug Mode*: Permite hacer un seguimiento de problemas de bus derivados del ruido etc, midiendo la calidad de las comunicaciones.

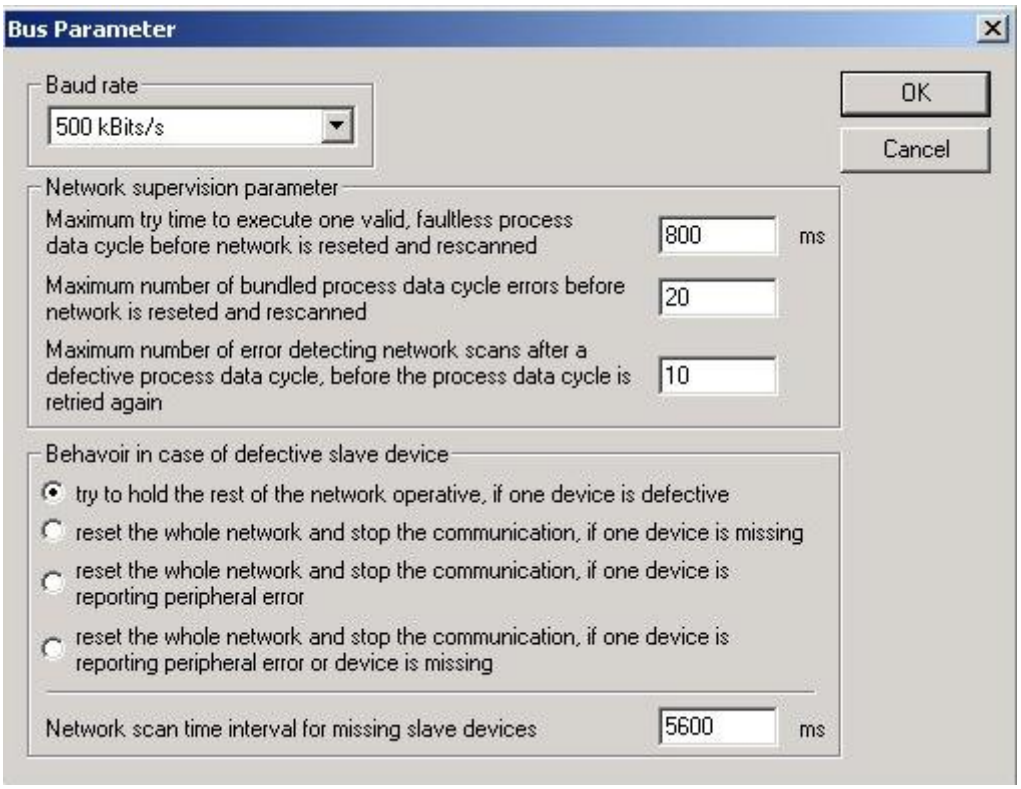
- *Download*: Descarga de la configuración de bus al Dispositivo.

Hay que hacer notar que hasta que no se descargue la configuración al dispositivo no conseguiremos que los datos que se han recopilado con *Automatic Network Scan* sean transferidos al sistema, con lo cual no serán utilizados por la tarjeta.

Estos datos de configuración de bus pueden ser archivados en un archivo con extensión .ib (en el caso de Interbus-S), .pb (en el caso de Profibus), etc. el cual puede ser importado desde la aplicación *Designer.*

Una vez configurado el bus y previo a hacer la descarga en la tarjeta controladora se deben realizar algunos ajustes de configuración más.

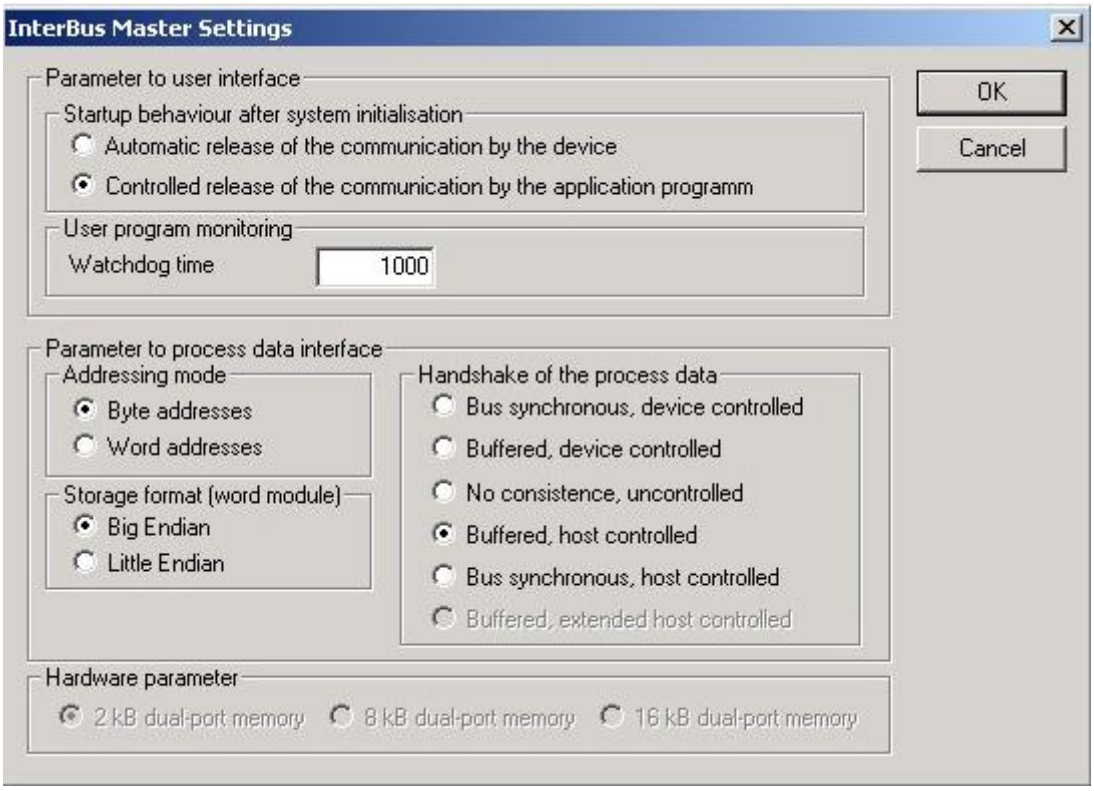
Primeramente se configurarán los parámetros de bus que indican como deseamos que reaccione el bus en caso de problemas. Esto depende del tipo de instalación, esta configuración se realiza desde las opciones de *Settings->Bus Parameter...* En el caso de Interbus, el diálogo que se muestra es el siguiente:



El tiempo etiquetado como *Maximum try time to execute one valid, faultless process data cycle before network is reseted and rescanned* está colocado por defecto a 800 mseg., este tiempo para muchas aplicaciones es muy alto y se recomienda bajarlo a 80 mseg.

Posteriormente se deben configurar los parámetros principales de la tarjeta controladora maestra. Para ello se seleccionará la opción *Settings->Master*

Settings... aquí se permitirán definir datos críticos para el correcto funcionamiento del sistema. A continuación se muestra la opción recomendada.



El tiempo de Watchdog a configurar depende de la instalación, pero no debería ser superior a 200 mseg, el tipo de handshake de los datos de proceso debe de ser en todo caso Buffered, host controlled.

A partir de este momento la configuración realizada debe archivarse y transferirse mediante la opción *Download* a la tarjeta de control.

1.1.1.1 Configuración específica para soportar servidor de reserva

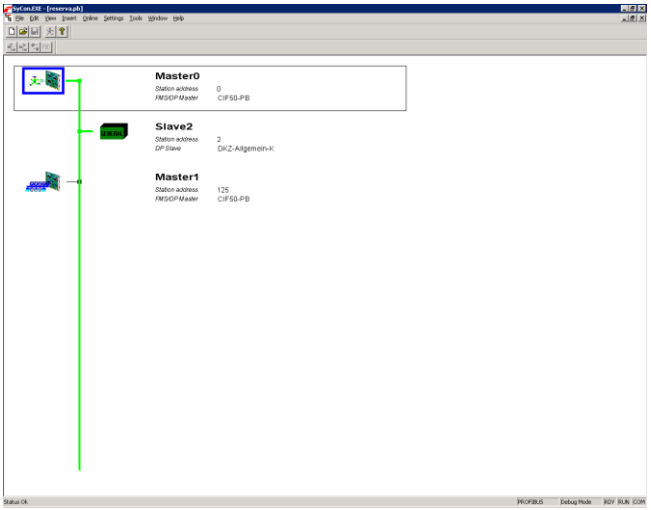
Con la salida de las versión 3.1 de Galileo y posteriores, se permite la utilización de otro computador como servidor de reserva, que tomará el control de la instalación en caso de errores en el servidor principal.

En este apartado se explica como configurar la tarjeta Hilscher para funcionar en modo reserva. Es importante tener en cuenta lo siguiente:

- Galileo 3.1 y posteriores puede funcionar en varios modos cuando trabaja como servidor de reserva. Estos modos se refieren a la manera en que se realiza el cambio de servidor en caso de error. El modo *Automático*, que cambia el controlador sin intervención del usuario, SOLO PUEDE SER UTILIZADO CUANDO LA TARJETA ES PROFIBUS. En el caso de Interbus este cambio tiene que hacerse de manera manual.

- En el caso del cambio automático en Profibus, es necesario que en la configuración del Sycon aparezcan ambas tarjetas, y una de ellas, la que inicialmente actuará como reserva, tenga una dirección un número menor que la que actúa como activa. Esto es, para el caso de que la tarjeta activa tenga una dirección 5, la de reserva necesita tener una dirección 4. En el caso habitual de que la tarjeta activa tenga la dirección 0, la de reserva tendrá la dirección 125.

La configuración básica del bus es la misma que en el caso de tener una única tarjeta. La diferencia está en que antes de cargar la configuración en la tarjeta es necesario añadir al bus la tarjeta de reserva, poniéndole como dirección la anterior a la de la tarjeta activa (en el caso de que la tarjeta activa sea la 0, la dirección de la de reserva es la 125). Una vez hecho esto, basta con enviar la configuración ambas tarjetas.



Es importante tener en cuanta que el fichero que se envía a las tarjetas ES EXACTAMENTE EL MISMO y que por tanto ambas tarjetas tendrán la misma dirección Profibus, lo que supone un problema ya que ambas entrarán en conflicto.

Es necesario un procedimiento para poner en marcha ambos sistemas de manera segura. Los pasos necesarios son los siguientes:

1. Configurar de manera correcta las tarjeras de cada computador. Para ello basta con realizar la configuración de manera tradicional y añadirle antes la tarjeta master de reserva una vez tengamos todos los dispositivos detectados. Una vez hecho esto se transferirá la configuración a cada tarjeta.
2. Configurar de manera correcta la consola de Galileo, indicando que PC será el de reserva y cual el activo.
3. INICIAR EN PRIMER LUGAR EL SERVIDOR DE RESERVA. De esta manera, si esta correctamente configurado Galileo, cambiará su dirección por la de reserva.
4. Iniciar el servidor activo.

1.2 Bus de campo INTERBUS-S

1.2.1 Introducción

Interbus es un sistema de bus serie para la transmisión de datos para diferentes sistemas de control (PLC's, Ordenadores, Robots) con entradas/salidas espacialmente distribuidas a las cuales se conectan los sensores.

Interbus se ha estandarizado con la norma DIN 19 258 y ha ganado gran aceptación en la automatización dado a que es una tecnología fácil y orientada al usuario final.

Interbus conecta módulos de entradas salidas distribuidas así como elementos de automatización (Variadores de frecuencia, encoders, etc.) para ser controlados por un ordenador por medio de transmisión de datos serie.

Interbus es un sistema abierto, lo que quiere decir que se pueden conectar elementos de diferentes fabricantes. Por lo tanto es el usuario el que debe elegir que tipo de dispositivos se adapta mejor a sus características.

1.2.2 La principal tarea de Interbus

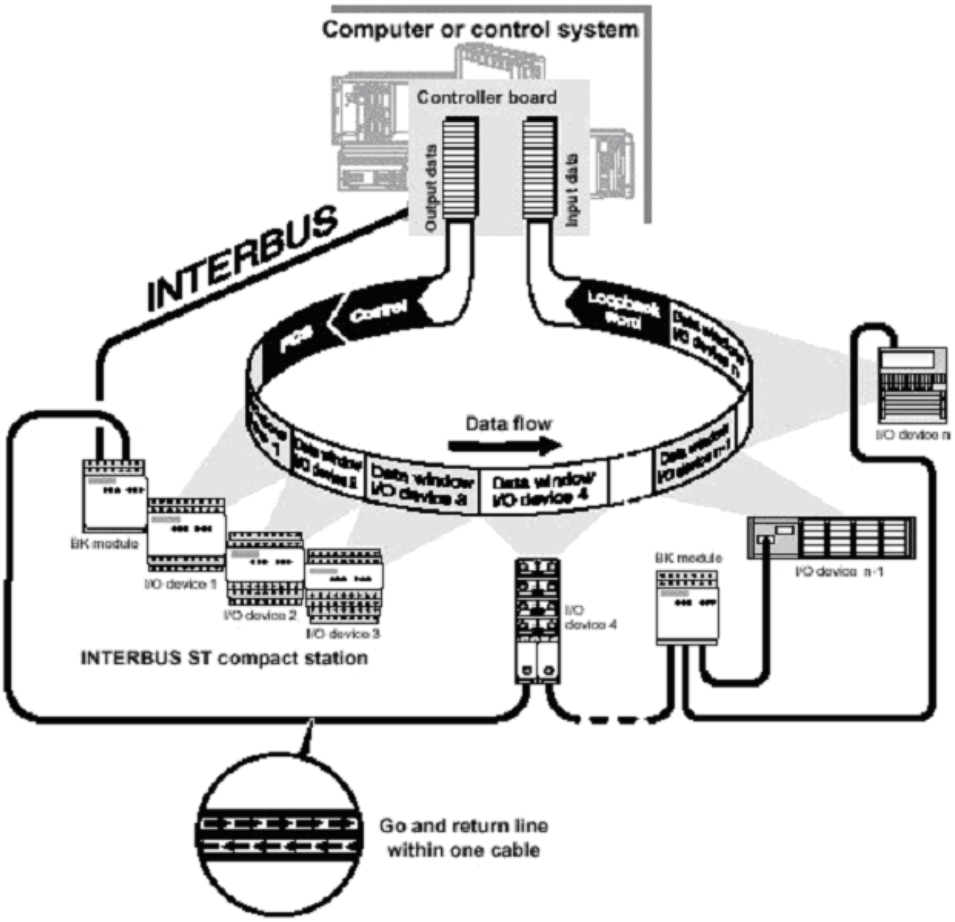
Interbus conecta señales de sensores / actuadores al elemento de control (PLC, PC, en general Host), cumpliendo dos tareas importantes:

- La transmisión cíclica de los estados de proceso rápidamente.
- La transmisión de datos de parámetros para módulos complejos de entrada / salida y dispositivos especializados que deban ser parametrizados durante la operación (Variadores de frecuencia, Robots).

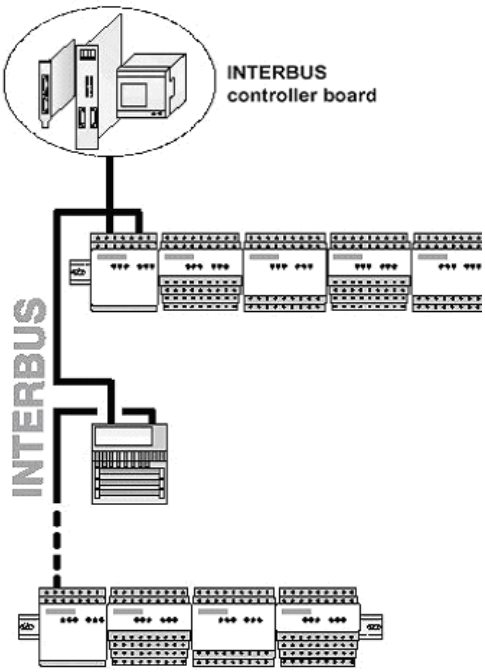
1.2.2.1 Estructura general del método de operación de Interbus

El sistema Interbus está diseñado como un anillo de datos. La tarjeta de control es el dispositivo central para controlar el anillo de datos. Intercambia datos transmitidos en serie por el anillo de datos entre el control de alto nivel del Host y los dispositivos de bajo nivel de Interbus. El intercambio de datos es realizado simultáneamente y cíclicamente en las dos direcciones (operación full-duplex).

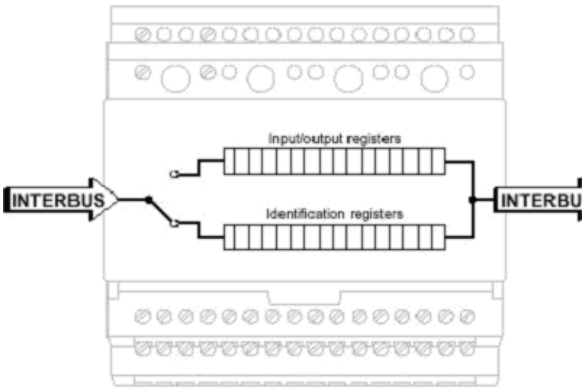
El anillo de datos tiene la estructura de un registro de desplazamiento distribuido de forma espacial. Todo dispositivo Interbus con función de entrada / salida conecta su periferia analógica o digital al anillo de datos del Sistema Interbus mediante sus registros de datos.



Para simplificar la instalación las líneas de ida y vuelta se implementan sobre una única línea de cable. El sistema permite bifurcar desde la línea principal, con lo cual la estructura que se tiene es de árbol.



Cada dispositivo en el Bus tiene un *registro de identificación* (Registro ID). El registro incluye información acerca del tipo de módulo, la longitud del registro en el anillo de datos, el estado actual y los estados de error. Adicionalmente los dispositivos de Entrada / salida tienen registros para la transmisión de datos de proceso.



Interbus diferencia entre dos tipos de ciclos:

- *Ciclo de Identificación* (ID Cycle) que se ejecuta para inicializar el sistema Interbus o bien a petición del Host. En este ciclo se lee el registro ID de todos los dispositivos y a partir de este momento el sistema genera la imagen de proceso. Normalmente el ciclo de identificación ocurre al arrancar el sistema, esto permite averiguar la configuración de bus, compararla con la última conocida. Después de un ciclo de ID realizado con éxito los dispositivos de Interbus son conmutados internamente a los registros de datos y solamente se transmiten ciclos de datos.

- Los *Ciclos de Datos* son los responsables de la transmisión de datos. En un ciclo de datos la controladora del Host actualiza las entradas salidas de todos los dispositivos Interbus a la vez.

La longitud de registro mediante la cual un dispositivo se acopla al anillo de datos sólo depende de la longitud de sus registros de entrada/salida. El registro de ID tiene una longitud fija de 16 bits y no se toma en cuenta.
Obviamente Interbus realiza un continuo chequeo del flujo de transmisión y sus datos para verificar que todo es correcto:

- Si los datos han sido transmitidos de forma apropiada, la entrada de datos será aceptada por la tarjeta controladora y los datos de salida transmitidos a los dispositivos.
- Si se detecta un error, los datos del ciclo con fallo se ignoran ya que la corrección de estos datos llevaría más tiempo que la ejecución de un nuevo ciclo.

1.2.3 Protección de la transmisión Interbus

Interbus permite la transmisión segura de datos bajo condiciones industriales mediante los siguientes métodos de protección:

1.2.3.1 Transmisión de señal diferencial

Se utiliza un cable apantallado de pares trenzados acorde a la norma RS485. Los datos de usuario se transmiten como señales diferenciales sobre las líneas de un par trenzado.

1.2.3.2 Chequeo del Bus remoto

La conexión entre dos dispositivos se chequea intercambiando una información especial de estado. La ruptura del cable de bus o un conector que no haga buen contacto produce el consiguiente mensaje de error como resultado del chequeo de Bus.

1.2.3.3 Chequeo del bucle

La tarjeta controladora registra en la palabra de retorno de bucle el estado de la comunicación. Esta palabra se marca en el momento exacto de la partida de los datos, pasa por todos los dispositivos del bus y retorna al final de la imagen de proceso de entradas de vuelta a la controladora. De este modo puede verificar si el tiempo requerido para realizar la transmisión de datos ha sido correcto y se registra mediante una secuencia de bits si ha habido problemas en la ruta de transmisión o en algún dispositivo.

1.2.3.4 CRC

CRC (Chequeo de Redundancia Cíclica) se comprueba en cada transmisión entre dos dispositivos Interbus. Para esto, se genera una palabra de CRC en cada dispositivo y en la tarjeta controladora maestra.
Cuando se termina un ciclo de transmisión se chequea la palabra de test para comprobar la consistencia de los datos recibidos. Si se produce una inconsistencia,

se indicará a la tarjeta controladora a través de todos los dispositivos y se iniciará un nuevo ciclo.

1.2.4 Rutina de Test

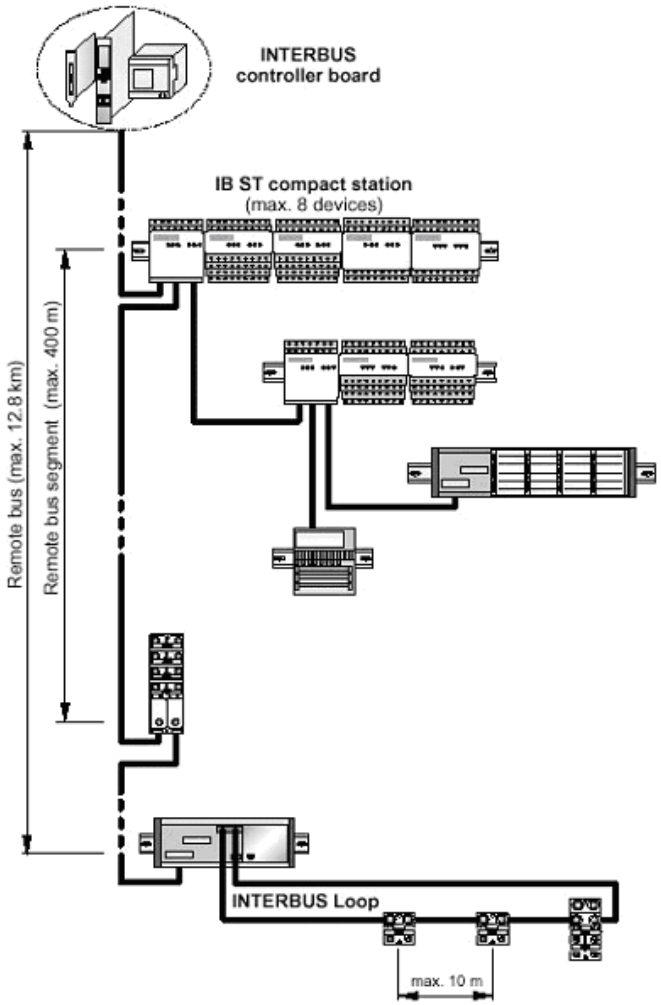
Una vez que se arranca todo el sistema ocurre la siguiente secuencia de inicialización:

- Se chequea la tarjeta controladora
- Se chequea la circuitería lógica y los componentes de Interbus.

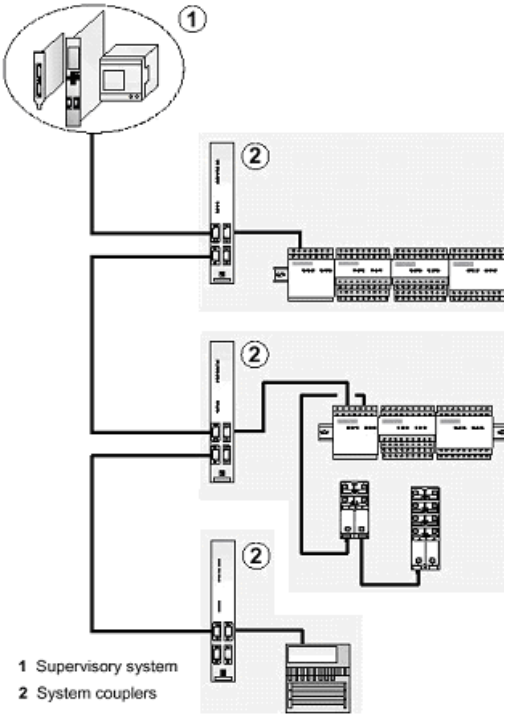
Los elementos conectados a Interbus se inicializan, para esto la tarjeta controladora activa todo el Interbus empezando un segmento después de otro y chequeando errores.
Primero la tarjeta controladora activa la conexión hasta el primer módulo terminal (BK) o el primer dispositivo de bus remoto. Si no ocurre ningún error, el bus local o el bus remoto conectado al BK se inicializan. Después de esto se inicializa la conexión con el primer elemento de bus remoto, etc.

1.2.5 Topología de Interbus

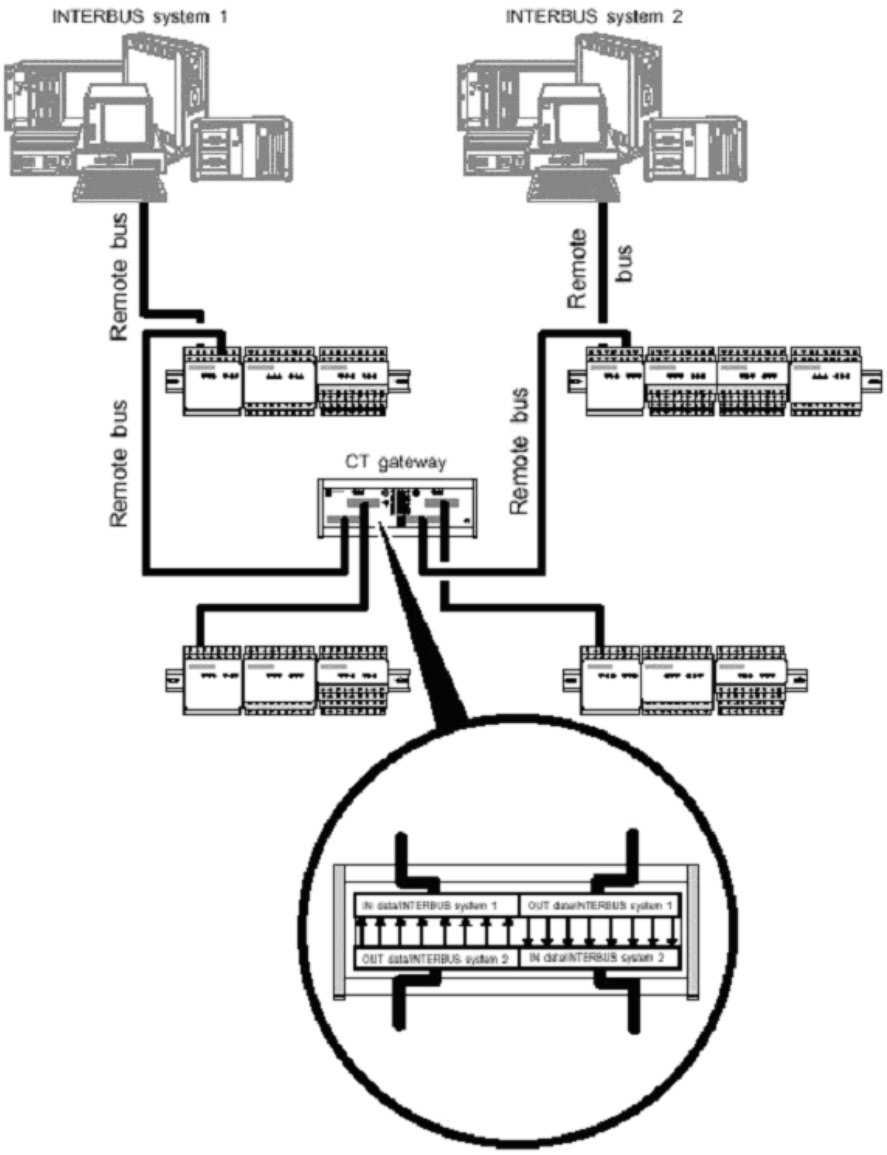
En la siguiente ilustración se visualizan los límites de conexionado de Interbus.



Interbus dispone la posibilidad de conectar sistemas supervisores de más alto nivel (posibilidad todavía no experimentada por MECALUX)

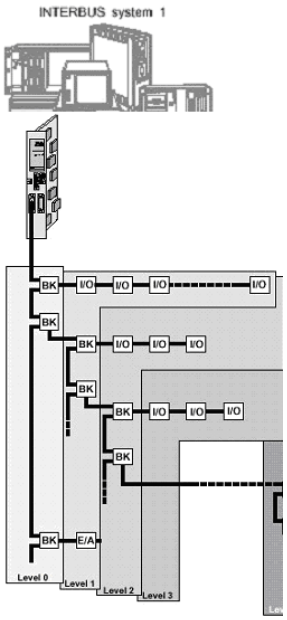


Y otra posibilidad muy interesante son los sistemas Gateway que permiten unir buses independientes.

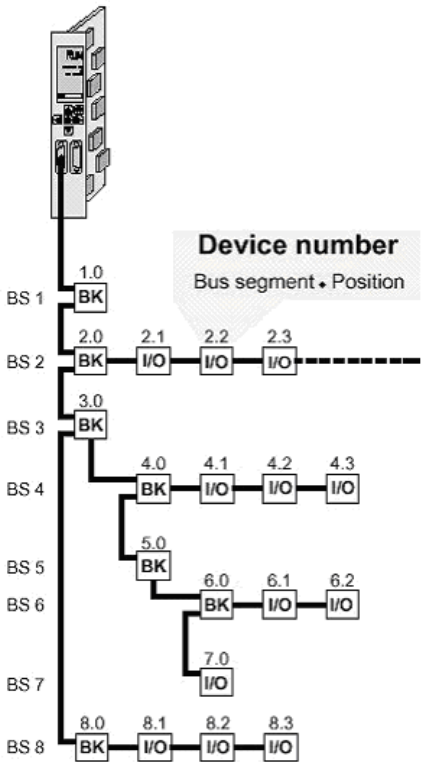


1.2.5.1 Topología de Bus Remoto

Básicamente en nuestras instalaciones utilizamos topologías de bus remoto. Estas topologías permiten disponer de 16 niveles de bus remoto a través de bifurcaciones (en el caso de Phoenix IBS ST 24 BK RB-T) haciendo posible bifurcar hacia el siguiente nivel de bus. Este sería un ejemplo de bifurcación y hasta que límites se puede llevar.



Esta forma de estructurar el bus sirve para ahorrar cableado, al ser un bus serie si no existiera una posible configuración en árbol se desperdiciaría muchísimo cableado.
Debe quedar absolutamente claro la estructura numérica que se define para referirse a los diferentes elementos de bus. En Interbus los elementos se identifican con dos números separados por un punto, de la forma *Segmento.Posición* continuamente se ven diagramas de bus en la documentación de Phoenix que representan buses donde siempre tenemos algún bus local, lo cual haría aparecer en el mismo elementos con Posición distinto de 0, esto no es normal en las instalaciones de MECALUX ya que la estructura de bus que se utiliza es de bus remoto, con lo cual todos los elementos serán del tipo *Segmento.0*. El siguiente ejemplo muestra una estructura de bus con los dos tipos de elementos.



1.2.6 Tiempo de ciclo de bus

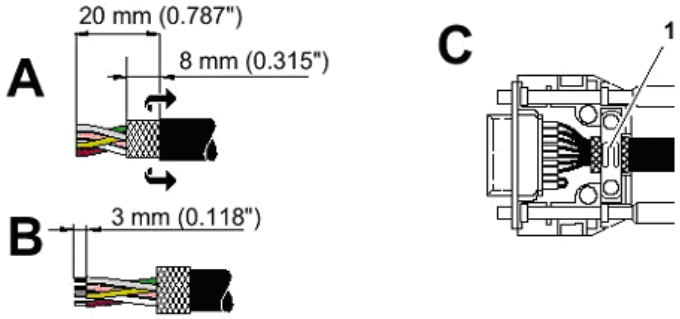
El tiempo de ciclo de bus en Interbus es fijo y constante. Se puede realizar un cálculo teórico del mismo a través de la siguiente fórmula matemática:

$$t_t = (1.15 * 13 * (8+n) + 3a) * t_b + t_s + 2t_p$$

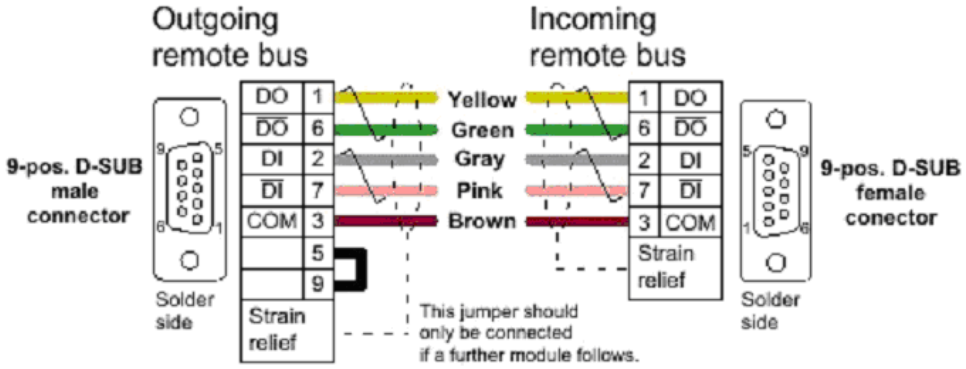
t_t	Tiempo de Transmisión en milisegundos
n	Número de bytes de datos (datos de proceso + PCP)
a	Número de dispositivos de bus
t_b	Duración de Bit = 0.002 ms
t_s	Software runtime: 0.34 ms
t_p	Cable = 0.016 ms/km

1.2.7 Descripción del conector típico de Interbus

A continuación se describe el conector típico de Interbus y su correcto conexionado.



El cableado debe realizarse de la siguiente manera:



NOTA:
Los siguientes problemas son típicos y deben ser tenidos en cuenta:

- *Interbus detecta que detrás de un módulo hay otro porque el conector de salida del primero tiene un puente realizado entre los pines 5 y 9. Sin este puente, la detección acabaría en el módulo en cuya salida se ha omitido el puente.*
- *Si el cableado no es correcto no hay detección posible lo más inteligente es ir desconectando el cable de salida de los diferentes elementos y realizando detecciones para ver donde se encuentra el error de cableado.*

1.2.8 Instalación y configuración de la tarjeta Hilscher

La instalación y configuración de la tarjeta ya ha sido descrita en el apartado anterior genérico para todas las tarjetas utilizadas, independientemente del modelo.

1.3 Bus de campo PROFIBUS-DP

1.3.1 Introducción

PROFIBUS define las características técnicas y funcionales de un bus de campo serie con el cual controladores programables digitales distribuidos que pueden ser repartidos desde el nivel del sensor/actuador a través del nivel de campo y hasta el nivel de célula. PROFIBUS distingue entre unidades maestras (master) y esclavas (slave).

La estandarización que define PROFIBUS es IEC 61158 e IEC 61784, que los definen como un estándar de bus de campo abierto e independiente de fabricante. Esto permite a dispositivos de diferentes fabricantes ser usados en red sin necesidad de adaptadores, y lo hace adecuado para tareas de respuesta crítica y también para tareas más complejas de comunicación.

En PROFIBUS, las señales de los sensores binarios y actuadores son transferidas de forma cíclica a nivel de sensor. La periferia distribuida, como módulos de entrada/salida transmisores, actuadores, válvulas o terminales de operador se comunican a nivel de campo mediante un interfaz de comunicación en tiempo real con los controladores programables del nivel de célula. La imagen de proceso es generalmente, transferida de forma cíclica.

Los controladores programables (PLCs), se comunican entre si a nivel de célula. El flujo de información necesita paquetes de datos de gran tamaño y funciones de comunicación potentes.

1.3.2 Aspectos generales

1.3.2.1 Principio Master-Slave

Los Master determinan la comunicación de data en PROFIBUS. Un Master puede enviar mensajes sin necesitad de que una señal externa los provoque, siempre que tenga autorización de acceso al bus. Los Masters son designados también como estaciones activas en el protocolo PROFIBUS.

Un mensaje enviado por una estación master puede estar destinado a una estación individual, a un grupo de estaciones o a todas las estaciones. Solo los mensajes individuales son confirmados de forma inmediata con un ACK enviado por el bus hacia el emisor.

1.3.2.2 Principio del TOKEN

La autorización de acceso al bus (el token) es pasada transcurrido un cierto tiempo, a través del bus, a un nuevo master, de forma que se permita a todos los master acceder al bus. Esto resulta en un anillo token-ring lógico entre las unidades master. Cada master puede manejar datos y comunicación durante el tiempo que tenga el token.

Los esclavos no reciben autorización de acceso al bus. Son estaciones pasivas. Sólo envían ACK sobre los mensajes que reciben en respuesta a las peticiones del

master. Cada slave es asignado a un master. El master puede direccionar a sus slaves durante el tiempo que tenga el token.

Existen varias posibilidades de configuraciones: redes con solo master, combinaciones de masters y slaves y también redes con un solo master, llamados sistemas mono-master. Este último tipo de comunicaciones master-slave es óptimo para sistemas de respuesta crítica.

Mientras tenga el token, el master intercambia datos con sus slaves. Este sistema de polling (encuesta), es realizado de forma sucesiva en el orden en que los slaves han sido dados de alta en el bus o en el orden en el cual han sido activados. Este procedimiento es repetido cíclicamente, pero es interrumpido cuando el token cambia de master.

1.3.2.3 Tecnología de transmisión

La transmisión por cable de acuerdo con los estándars EIA RS 485 es el tipo de transmisión usada por este protocolo. Usa un cable de par trenzado de cobre con un par de conductores. Las guías de fibra óptica hechas de cristal o plástico están disponibles para entornos sujetos a interferencias mayores o por necesidades de aumentar el rango de alcance.

La transmisión se transmite de forma serie. Cada byte de información es formateado por el RS 485 añadiéndole un bit de comienzo, un bit de stop y un bit de paridad. Es decir, que se transmiten 11 bits por el cable por cada byte de información (8 bits). Esto proporciona una buena protección contra errores de transmisión.

Cuando se cablea una red PROFIBUS, cada sección de bus (segmento) debe ser terminado con una resistencia de terminación que esta constantemente alimentada. Esto puede realizarse mediante jumpers en el conector o por elementos de terminación activos.

Una red PROFIBUS puede consistir en varios segmentos interconectados mediante repetidores. Los repetidores amplifican la señal y son necesarios si hay más de 32 estaciones en el bus. Se pueden conectar hasta 3 repetidores entre 2 estaciones comunicándose unos con los otros sin refresco de señal, o hasta un máximo de 9 repetidores con refresco de señal. Debe notarse que cuantos más repetidores se añadan, más se incrementará el tiempo de transmisión necesario. Esto debe tenerse en cuenta para al caso de estructuras de bus muy largas. Se pueden implementar topologías de línea, árbol o estrella usando repetidores que incorporen tecnología 485.

Cables de fibra óptica puede ser usada en lugar de los cables de cobre en entornos sujetos a fuertes interferencias electromagnéticas, para aislarlas eléctricamente o para poder extender la distancia a la que se encuentran las estaciones, independientemente de la velocidad de transmisión. Las estaciones PROFIBUS de fibra óptica pueden consistir en estaciones con interfaces ópticos directamente interconectados en una topología de línea. Las estaciones sin interface óptico pueden conectarse con sus interfaces RS 485 usando un terminal de bus óptico (OBT).

1.3.2.4 Velocidad de transmisión

La velocidad de transmisión puede ser establecida en pasos de 9.6 kbits/s hasta un máximo de 12 Mbit/s en el caso de RS 485, depende de la máxima extensión del segmento más largo y de las características físicas de la estación más "débil" en la configuración PROFIBUS. En el caso de la Fibra óptica, se puede cubrir el espectro entero de velocidades de transmisión, aunque debe tenerse en cuenta las características de los componentes ópticos activos usados.

1.3.2.5 Longitud de bus

El rango máximo de un segmento va en función de la velocidad de transmisión deseada, que debe ser la misma en todos los segmentos de una red PROFIBUS.

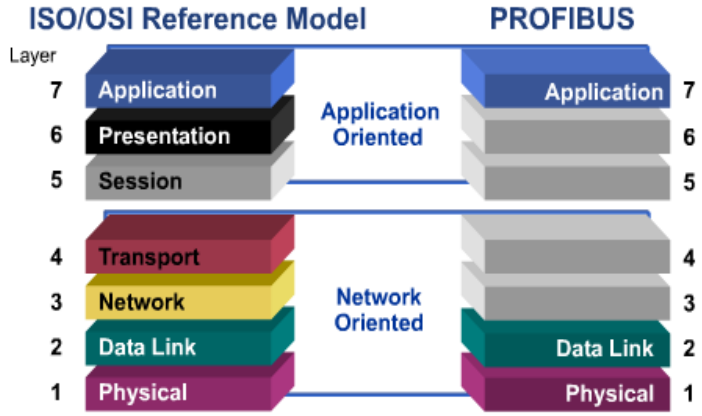
Velocidad de transmisión (kBits/s)	Distancia (metros)
12000	100
1500	200
500	400
187.5	1000
93.75	1200
19.2	1200
9.6	1200

La extensión máxima de la red se consigue conectando segmentos de la longitud máxima juntos. El número de segmentos (es decir, el numero de repetidores en series) depende del tipo de repetidor. En caso del RS 485, es posible conectar hasta 3 en serie y si es con refresco de señal, hasta 9 también se pueden usar módulos de fibra óptica para hacer de puente entre segmentos de RS 486 más separados.

Cada estación con un interface RS 486 en PROFIBUS representa una carga de bus. Se pueden conectar hasta un máximo de 32 estaciones de este tipo a un segmento PROFIBUS.
Sólo las estaciones Master o slave tienen direcciones de estación en PROFIBUS. Los repetidores u otros módulos acoplados no las tienen. Hasta un máximo de 127 estaciones, controladores programables, dispositivos de campo o unidades de programación pueden ser conectadas y direccionadas individualmente en una red PROFIBUS sobre varios segmentos (las direcciones varían entre 0 y 126).
La dirección de recepción 127 en cada estación está reservada para mensajes dirigidos a cualquier estación o grupo de estaciones en los llamados modos broadcast o multicast de mensajes. Los receptores de este tipo de mensajes no envían ACK.

1.3.3 Arquitectura del protocolo

La arquitectura del protocolo PROFIBUS está basada en las 7 capas del modelo ISO, que es un estándar internacional para comunicaciones. Las 7 capas de ISO definen los correspondientes servicios o reglas de ejecución para la comunicación entre 2 aplicaciones. Hay capas orientadas a estación, y capas orientadas a red.



A nivel de la capa física, se define el estándar EIA RS 485, aunque también es posible usar fibra óptica.

La capa de enlace de datos realiza las funciones de control de acceso al bus, integridad de datos y manejo de los protocolos de transmisión en PROFIBUS, llamada Fielbus Data Link (FDL) permitiendo enviar y recibir los mensajes a través del bus.

Las capas 3 a 6 de ISO no son distintivas en PROFIBUS, es decir, que la funcionalidad que proporcionan no es requerida por PROFIBUS.
La última capa, la de aplicación, tampoco es distintiva. Esto permite que la arquitectura se comporte de forma eficiente y rápida para la transmisión de datos. El acceso directo a las funciones de la capa 2 es establecido por el Direct Data Link Mapper (DDLM) como interface de usuario en PROFIBUS DP. En el caso de PROFIBUS FMS (universal), la capa 7 consiste en el Fieldbus Message Specification (FMS).

El control de acceso al Bus (MAC, Medium Access Control) define cuando una estación del bus puede enviar datos. Debe asegurarse que solo una estación tiene esta autorización en un momento dado. Para ello se usa el paso de token y el sistema de Master-Slave.

Una tarea importante de la capa 2 es la seguridad de datos. Puesto que la corrupción en la transmisión de datos es fácilmente detectable gracias a la recepción de ACK, añadido de bits y bytes de chequeo, los telegramas afectados son repetidos automáticamente en el caso de error.

Para la transmisión de datos vía PROFIBUS, la capa 2 define 3 servicios de transmisión, cada uno de los cuales permite los diferentes requerimientos:

- El servicio SDA envía datos a estaciones individuales y requiere a la estación que devuelva un telegrama de reconocimiento (ACK).
- El servicio SRD envía datos a estaciones individuales y a la vez requiere a la estación que también envíe datos. La estación responde con un telegrama de reconocimiento (ACK) y a la vez envía los datos requeridos sin tener que realizar otro acceso al bus. Este servicio es usado en la comunicación Master-Slave.

- El servicio SDN envía datos a una o más estaciones sin necesidad de que estas le respondan con un telegrama de confirmación. Este servicio es usado para enviar telegramas broadcast o multicast.

1.3.3.1 Paso del token

Todos los masters del bus deben compartir el tiempo de acceso al bus. Con sistema de paso de token, se convierte en un anillo Token-Ring lógico, en donde el derecho de acceso al bus se va pasado de un master al siguiente. Este orden se basa en que la dirección de las estaciones es una secuencia numérica ascendente.

Cada Master determina de forma independiente la lista de estaciones activas (LAS) en el bus y anota su propia dirección de bus (This Station: TS), la dirección de la siguiente estación (Next Station: NS) y la que le precede (Previous Station: PS). El telegrama del token es siempre enviado a la dirección NS y recibido desde la PS.

Cuando se produce un fallo en la configuración o en una estación que provoca un hueco en la lista de direcciones, cada estación crea una GAP LIST (lista de huecos). La estación comprueba las direcciones en esta lista cuando tiene el token, y si el tiempo de transmisión todavía esta disponible en el status de consultas del FDL. Consecuentemente, se deben evitar los huecos en la numeración entre estaciones, ya que se optimiza la circulación del token.

Cada Master mide continuamente el tiempo de circulación del token entre dos recepciones del mismo. La diferencia con respecto al tiempo de circulación planeado es el tiempo de acceso al bus que es el máximo tiempo disponible para el Master. El tiempo que no lo requiera, puede ser usado automáticamente por otros Master.
Básicamente, cada master siempre puede manejar una secuencia de mensajes de alta prioridad para la recepción del token, independientemente de si actualmente todavía le queda tiempo de acceso al bus de acuerdo a la medición de tiempo de circulación del token.

Si una estación activa falla, o una nueva estación activa es añadida, el token ring lógico es auto-reconfigurado sin interrumpir el intercambio de datos en el bus.
Se puede imponer un límite máximo que a la lista de posibles direcciones de estación. Se define como el parámetro de configuración HSA (High Station Address). Todas las estaciones del anillo token ring lógico deben tener un número menor que este parámetro.

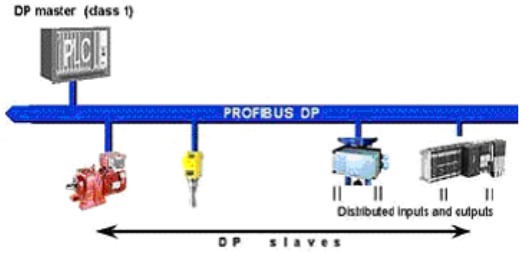
El telegrama token consiste en un start byte (0xDC en hexadecimal), la estación de destino (DA=NS) y la estación de origen (SA=TS). Este telegrama no requiere notificación de recepción, sin embargo, el emisor comprueba que el receptor está manejando activamente el bus. Si este no es el caso, el token es enviado a la siguiente estación del bus tras dos reintentos. Si sólo hay un Master en el anillo, este se envía el token a si mismo. Tales sistemas, conocidos como mono-masters, son óptimos para comunicaciones rápidas con periferia distribuida, es por eso por lo que es la configuración más frecuente en el caso de PROFIBUS DP.

Lógicamente, la comunicación master-slave esta sujeta a la circulación del token. Si un master tiene autorización de acceso al bus (tiene el token), tiene opción de enviar mensajes a sus slaves, o obtener mensajes de ellos. Los slaves pasivos no tienen opción de enviar mensajes independientemente.

Se puede realizar direccionamiento adicional (DAE, SAE), como parte de la unidad DATA, gracias a una identificación (el bit más significativo vale 1) en la dirección del emisor (SA), o en la dirección del receptor (DA). Esto permite a una estación especificar áreas distintas de emisión o de recepción.

1.3.3.2 Perfil de comunicaciones DP

PROFIBUS DP (Periferia Distribuida), es una variante del protocolo optimizada para velocidad y bajo costo de las comunicaciones entre un DP master y su periferia distribuida, los DP slaves repartidos por el bus. Alcanza su mejor rendimiento en sistemas mono-master sin restricciones de acceso al bus. Esto es debido a que el controlador central requiere un ciclo de acceso al bus (tiempo en el que todos los esclavos son preguntados a la vez) que es menor que el ciclo de proceso interno del sistema de control.



En el caso de los sistemas multi-master, el controlador central deben compartir el acceso al bus. Esto, frecuentemente resulta en unos requerimientos más restringidos en cuanto a la velocidad de transmisión o el volumen de datos manejado, a fin de poder asegurar el tiempo de ciclo de bus requerido. Los DP Masters no se intercomunican a través del protocolo DP.

Hay que realizar una distinción entre dos clases en el caso de un DP Master:

1. *Clase 1* (DPM1): típicamente es un PLC o sistema de control central, con ciclos estipulados y constantes de intercambio de información con las estaciones slave distribuidas.

2. *Clase 2* (DPM2): estos son dispositivos más especiales de ingeniería, operación o configuración. Sólo se usan si son necesarios, por ejemplo para diagnosis. UN DPM2 puede configurar los dispositivos conectados a través del bus, evaluar los valores medidos y los parámetros de los dispositivos, y preguntar por el estado de los dispositivos.

El comportamiento estándar de un sistema en el caso de PROFIBUS DP esta esencialmente determinado por el estado operativo del DPM1. Este puede ser controlado por una unidad de configuración ya sea local o a través del bus. Se puede distinguir 3 estados en los que puede estar: Stop, Clear y Operativo.

- Estado *Stop*, no hay comunicación de datos entre DPM1 y los slaves.
- Estado *Clear*, el DPM1 lee la información de las entradas de los slaves y mantiene las salidas de los slaves en un estado seguro.

- Estado *Operativo*, el DPM1 esta en fase de transferencia de datos. En la comunicación de datos cíclica, las entradas son leídas desde los slaves y la información de salidas es enviada a los slaves.

Antes de que comience el intercambio cíclico de datos, existe una inicialización entre el DP master y los DP slave. Esto implica comprobar que la configuración de los dispositivos corresponde con la esperada, por ejemplo en cuanto a formato y longitud de la información, número de entradas y salidas, etc.

En primer lugar, el DP master envía una petición de diagnostico al DP slave. La respuesta de diagnostico contiene el estado de la estación, la dirección PROFIBUS del DP master en el que el slave fue programado, el identificador del fabricante y datos específicos del dispositivo. Si la diagnosis del slave indica que necesita ser programado y configurado, el sistema continua con un telegrama de parámetro de reconocimiento por el DP slave.

El siguiente paso tras la programación, consiste en que el DP master debe enviar la configuración al DP slave. Si el slave detecta diferencias con su configuración actual, genera la correspondiente información de diagnostico y se coloca en un modo en que no puede realizar comunicación de datos. En cualquier caso, el resultado del diagnóstico es enviado al DP master que decide si realizar más comprobaciones antes de empezar a realizar el intercambio cíclico de datos con el DP slave.

En modo normal de operación, el intercambio cíclico de datos ocurre entre el DP master y los DP slaves, se pueden enviar y recibir hasta 244 bytes de datos en un solo telegrama usando el servicio de SRD.

Si las salidas de varios DP slaves tienen que ser sincronizadas, estos esclavos pasan a modo sincronización mediante un comando para tal caso. Los datos de salida a transmitir consecutivamente a los slaves son almacenados en un buffer por los slaves y con colocados en las salidas sólo cuando reciben un comando de sincronización. De la misma manera, se pueden realizar también entradas sincronizadas mediante un comando enviado por el DM master. En ambos casos, la sincronización termina cuando el master lo decide enviando un comando apropiado. también existe la posibilidad de que los slaves con suficiente "inteligencia" se comuniquen entre sí. Estos pueden monitorizar las entradas de otros slaves a la vez que el DP master.

1.3.3.3 Equidistancia

Este término significa que el DP master comienza el ciclo de bus siempre cada cierto tiempo (mismo intervalo), asegurando de esta forma ciclos de bus de igual longitud. La equidistancia sólo es posible en sistemas mono-master con DPM1.

Los slaves, por lo tanto, reciben sus datos en intervalos constantes, lo cual es particularmente importante para dispositivos de posicionado o soluciones de control en bucle cerrado.

1.3.4 Instalación y configuración de la tarjeta Hilscher

Se ha descrito en un apartado propio la instalación de la tarjeta de control Hilscher utilizada CIF 50 PB.

1.4 Bus de campo CANOPEN

1.4.1 Definición general

El bus de campo CAN sólo define las capas físicas y de enlace por lo que es necesario definir cómo se asignan y utilizan los identificadores y datos de los mensajes CAN.

El protocolo CANopen, que está basado en CAN, e implementa la capa de aplicación propia de dicho bus de campo. La construcción de sistemas basados en CAN que garanticen la interconexión entre dispositivos de diferentes fabricantes necesita una capa de aplicación y unos perfiles que estandaricen la comunicación, la funcionalidad de los dispositivos y la administración del sistema:

- Capa de aplicación (CAL- *application layer*). Proporciona un conjunto de servicios y protocolos para los dispositivos de la red.
- Perfil de comunicación (*communication profile*). Define cómo configurar los dispositivos y los datos, y la forma de intercambiarlos entre ellos.
- Perfiles de dispositivos (*device profiles*). Añade funcionalidad específica a los dispositivos.

La capa de aplicación CAL está formada por cuatro servicios:

- *CMS* (*CAN-based Message Specification*): ofrece objetos de tipo variable, evento y dominio para diseñar y especificar cómo se accede a la funcionalidad de un dispositivo.
- *NMT* (*Network ManagemenT*): proporciona servicios para la gestión de la red. Realiza las tareas de inicializar, arrancar, parar o detectar fallos en los nodos.
- *DBT* (*DistriBuTor*): se encarga de asignar de forma dinámica los identificadores CAN, también llamados COB-ID (*Communication Object Identifier*).
- *LMT* (*Layer ManagemenT*): permite cambiar ciertos parámetros de las capas como por ejemplo el identificador de un nodo (Node-ID) o la velocidad del bus CAN.

Es importante tener en cuenta que en CANopen los parámetros de más de un *byte* se envían siempre en la forma *little endian*, es decir, primero el *byte* menos significativo (LSB).

El modelo de comunicaciones de CANopen define cuatro tipos de mensajes (objetos de comunicación):

- Objetos administrativos: son mensajes administrativos que permiten la configuración de las distintas capas de la red así como la inicialización, configuración y supervisión de la misma. Se basa en los servicios NMT, LMS (LSS) y DBT de la capa CAL.
- *Service Data Objects* (SDO): objetos o mensajes de servicio utilizados para leer y escribir cualquiera de las entradas del diccionario de objetos de un dispositivo. Corresponden a mensajes CAN de baja prioridad.

- *Process Data Objects* (PDO): objetos o mensajes de proceso utilizados para el intercambio de datos de proceso, es decir, datos de tiempo real. Por este motivo, típicamente corresponden a mensajes CAN de alta prioridad.
- Mensajes predefinidos: de sincronización, de emergencia y *time stamp*. Permiten la sincronización de los dispositivos (objetos SYNC) y generar notificaciones de emergencia en forma opcional.

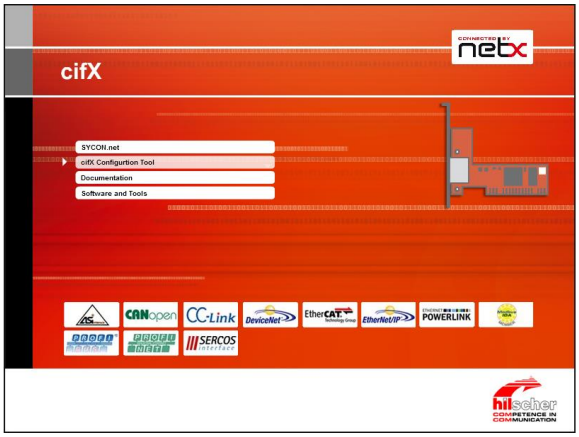
La tarjeta utilizada en Galileo para el control del bus de campo CANOPEN es de HIlscher CIFX-50, cuyo software de configuración se describe a continuación:

1.4.2 Instalación del software

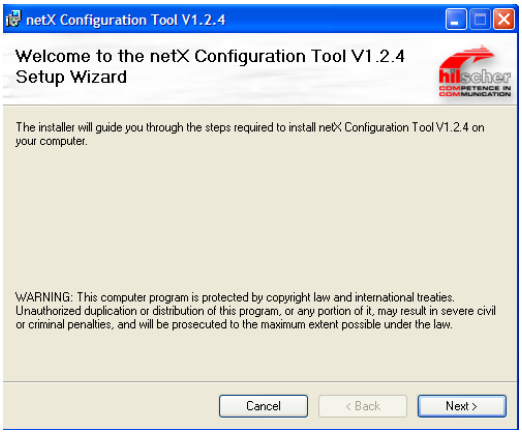
Lo primero de todo es insertar la tarjeta PCI en una de las ranuras del PC. Para ello, es necesario apagar previamente el PC, desmontar la carcasa de protección y alojar la tarjeta en el hueco especialmente diseñado para ello.

Una vez que la tarjeta se encuentra insertada en la ranura correspondiente es necesario instalar el software que se requiere para su configuración y diagnóstico. Dicho software está incluido en el CD que se suministra junto con la tarjeta. Al insertar el CD aparece la siguiente pantalla general que permite acceder a los distintos menús:

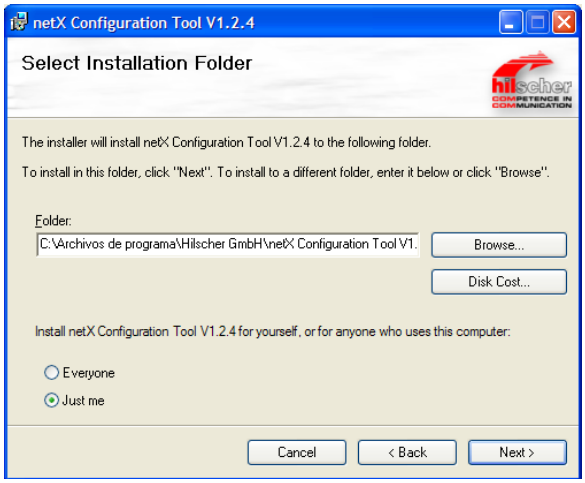
1.4.2.1 Instalar cifX Configuration Tool (software de diagnósito y configuración de la tarjeta Hilscher)



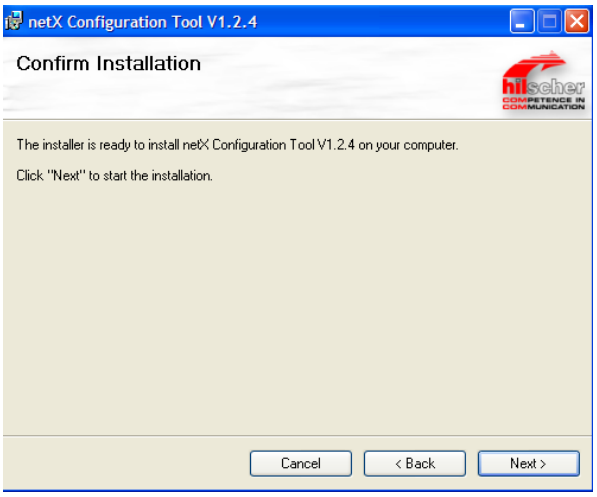
1. En la pantalla que aparece a continuación se ha de pulsar "*Next >*"



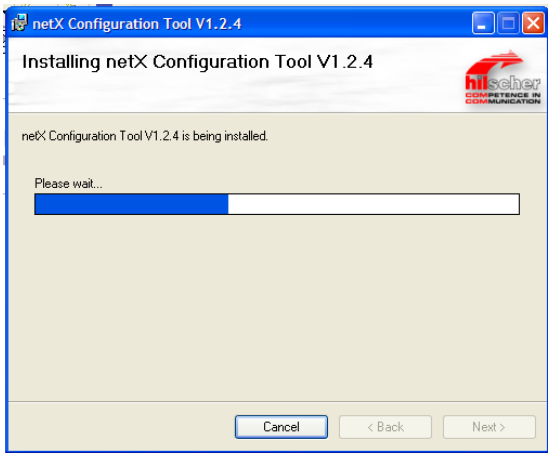
2. Se indica el directorio en el que se desea instalar la aplicación, se selecciona si la aplicación va a ser utilizada solo por el usuario actual o bien por todos los usuarios que entren en el PC (opción recomendada)y se pulsa de nuevo "*Next >*"

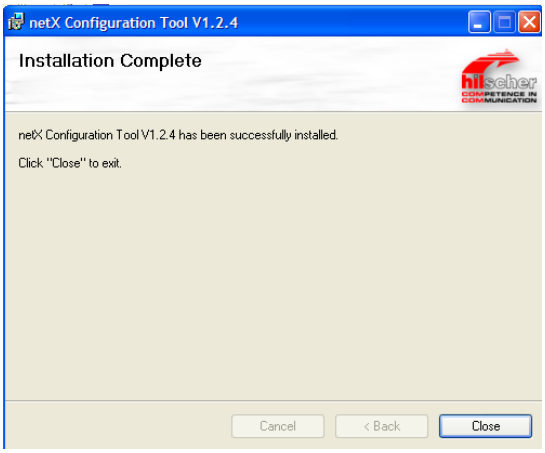


3. Se comienza la instalación de la aplicación

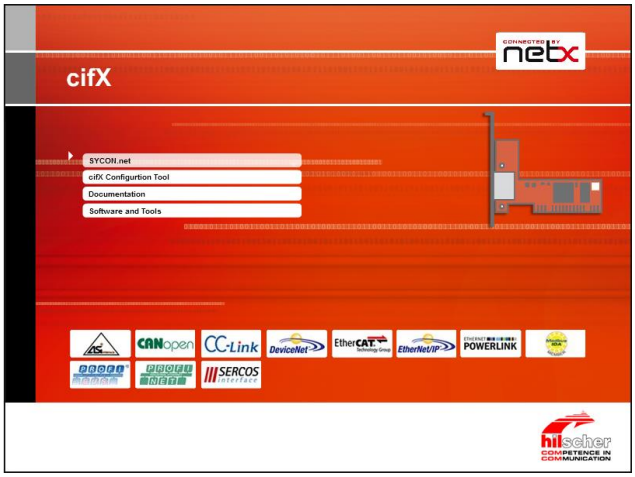


4. La instalación comenzará y cuando se finalice, se pulsará "*Close*"



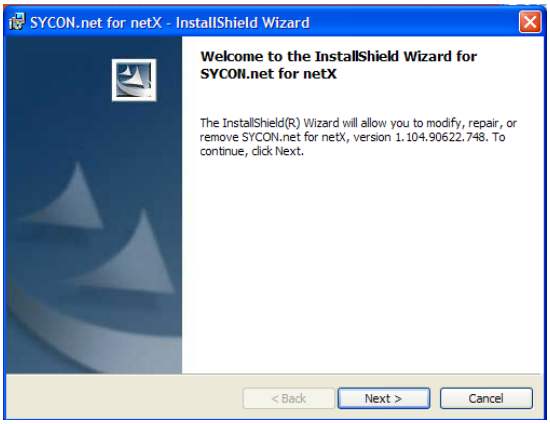


1.4.2.2 Instalar SYCON.net (software de diagnóstico y configuración de la tarjeta Hilscher y el bus de campo asociado)

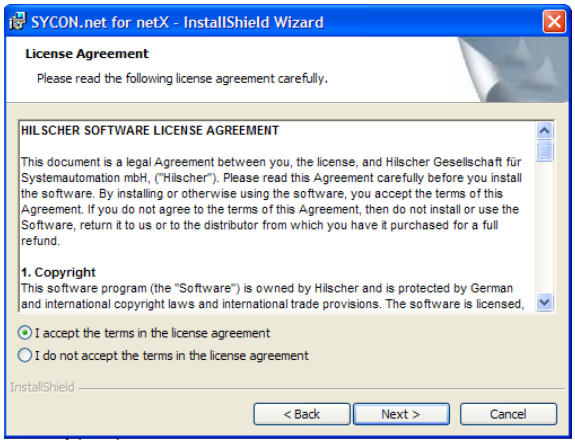


1. Inicialmente es necesario configurar el idioma deseado

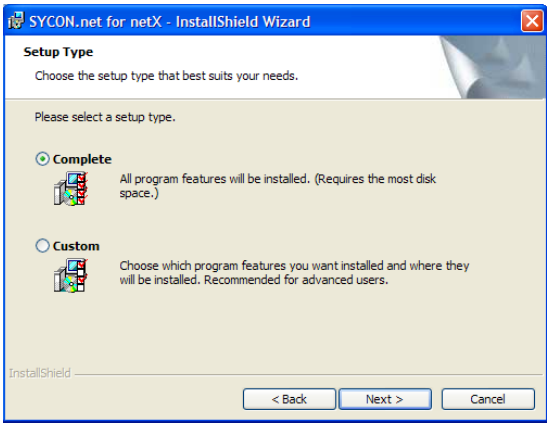
2. Comienza la instalación del software y se pulsa "*Next*" en la pantalla que aparece:



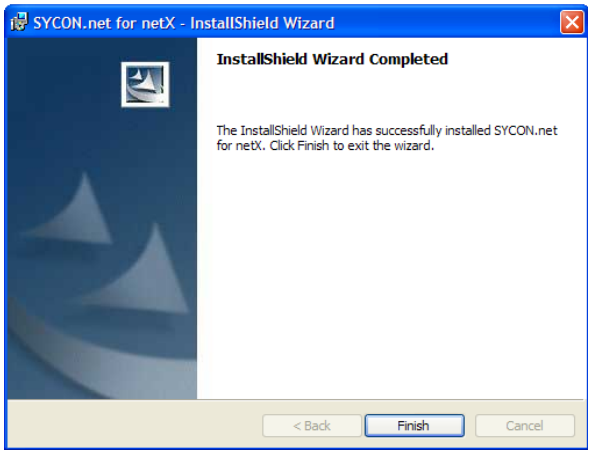
3. Una vez realizadas las comprobaciones necesarias, arranca la instalación real teniendo que aceptar las condiciones del software. Se configura también al igual que en netX, si el software será utilizado por el usuario actual o por cualquier otro usuario del PC:



4. Comienza la instalación del software, indicándole que ha de se de forma completa y no Custom:



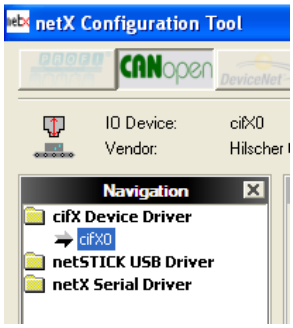
5. Cuando haya finalizado totalmente la instalación, se pulsa "Finish" para confirmar que ha terminado con éxito:



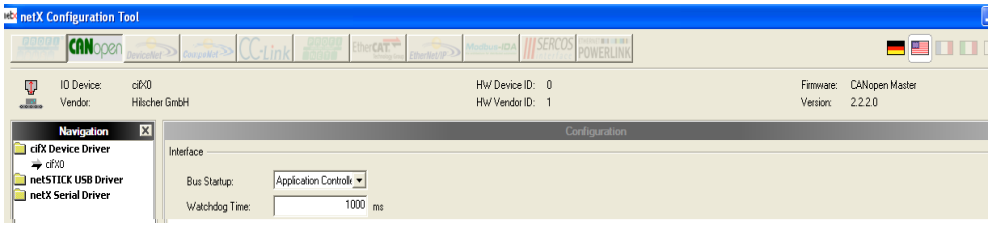
1.4.3 Parametrización, configuración y diagnóstico del bus de campo CANOPEN

1.4.3.1 netX Configuration Tool

Mediante dicho software podemos configurar y diagnosticar la tarjeta insertada, que aparecerá indicada en el árbol de la izquierda, y seleccionando en la parte superior el tipo de bus CANOPEN:

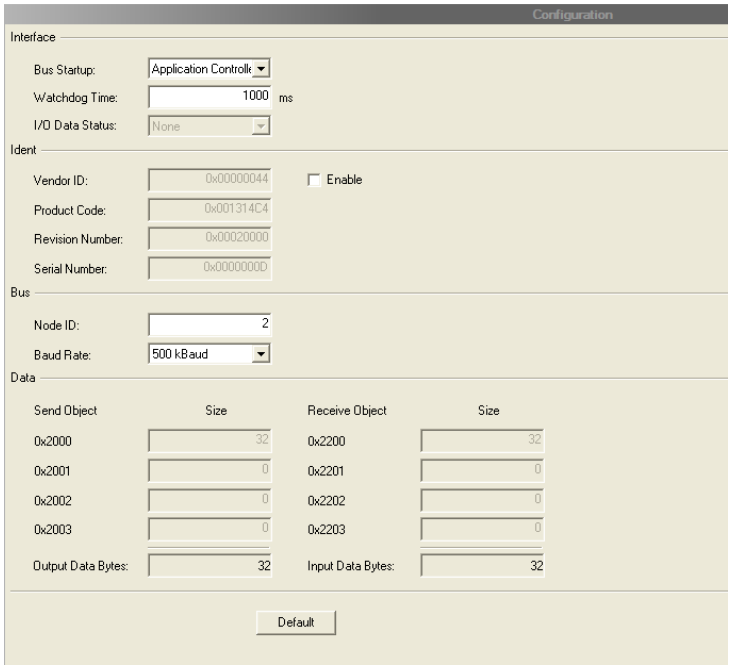


En la pantalla principal, se puede visualizar el tipo de tarjeta insertado, el ID Device, ID Vendor, versión y Firmware instalado. Además aparecerá otra información adicional, accediendo a ella mediante los botones de la parte inferior izquierda (Configuration, License y Diagnostic).



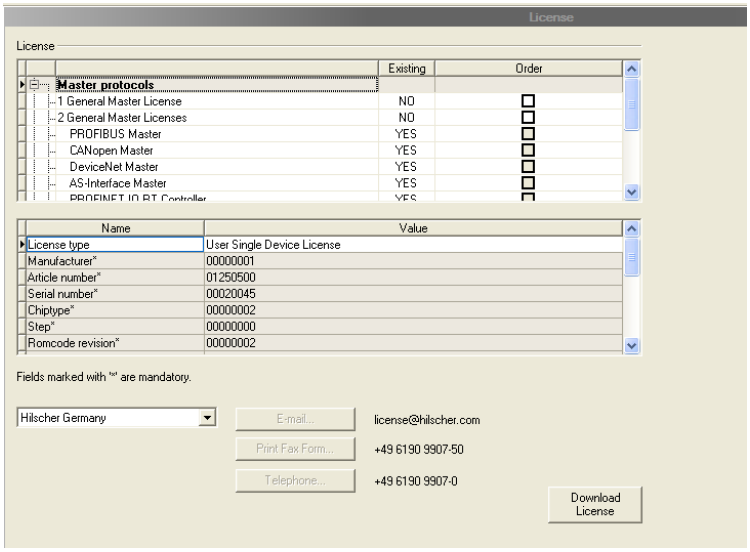
1. Configuration

Dentro de la pestaña configuración, aparecen distintos datos de configuración de la tarjeta, siendo por defecto los correspondientes al firmware instalado.
NOTA: Cualquier cambio en uno de estos datos hace que se descargue en la tarjeta el firmware correspondiente a CANOPEN Slave:



2. License

Aparecen las licencias instaladas de las que se dispone, para el tipo de funcionamiento y el bus de campo:

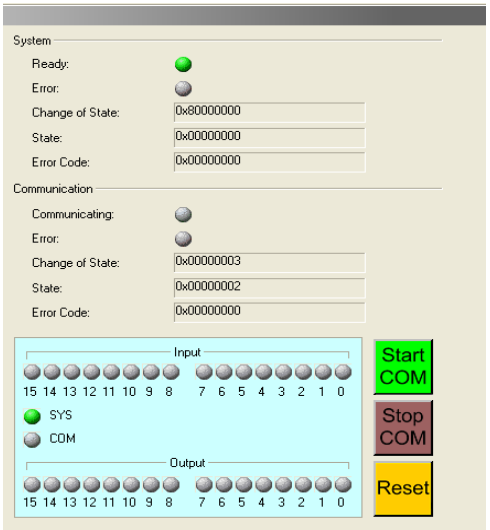


3. Diagnostic

Aparece un pequeño diagnóstico del estado de la tarjeta y la comunicación. Se pueden visualizar el estado de los dos primeros bytes de la periferia tanto de entradas como de salidas, así como forzar si se desea alguna de estas entradas.

Existen además tres opciones:

1. *Start COM* → Arranca la comunicación con el maestro
2. *Stop COM* → Detiene la comunicación con el maestro
3. *Reset* → Resetea la tarjeta.
   NOTA: Este botón hace que la tarjeta pierda el firmware instalado (sin pedir confirmación alguna) por lo que sería necesario volver a instalar el correspondiente al maestro.



1.4.3.2 SYCON.net

SYCON.net

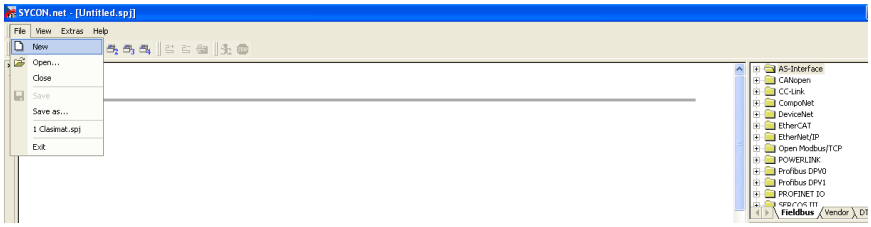
Este software permite realizar todas las tareas necesarias para trabajar con la tarjeta CIFX_CO y configurar el bus de campo asociado a ella:

- Establecer todos los elementos del bus de campo
- Configurar tanto la tarjeta master, como cada uno de los esclavos
- Evaluar y diagnosticar el estado de la comunicación y sus posibles errores

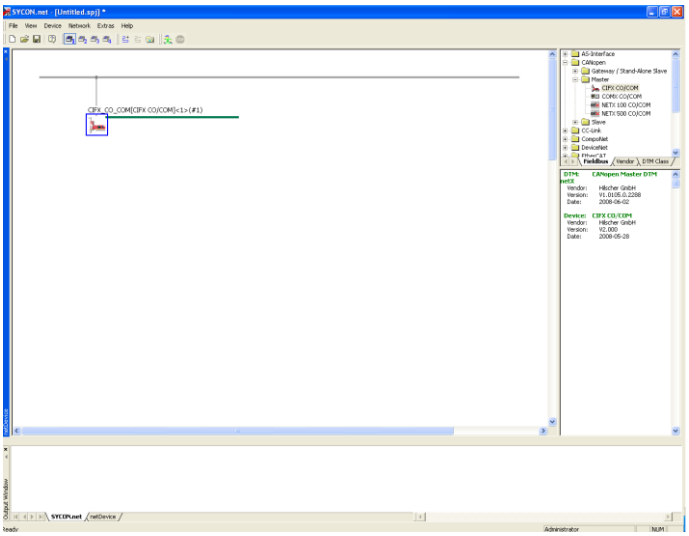
Se detalla a continuación los pasos a seguir para poder configurar el bus de campo CANOPEN de la aplicación:

1.4.3.2.1 Creación de un proyecto nuevo

Dentro del menú *File* → *New*



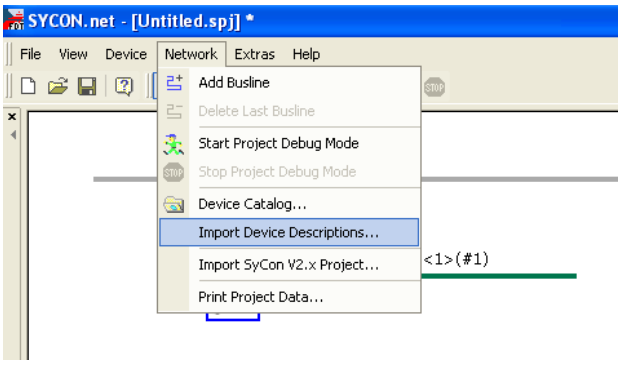
En la parte derecha aparecerá un árbol con los distintos buses de campo que se pueden utilizar, y una vez seleccionado CANOPEN, se escoge dentro de Master, la tarjeta a utilizar (CIFX_CO/COM). Se arrastra el icono de la tarjeta sobre la línea de bus que aparece por defecto en la pantalla general.



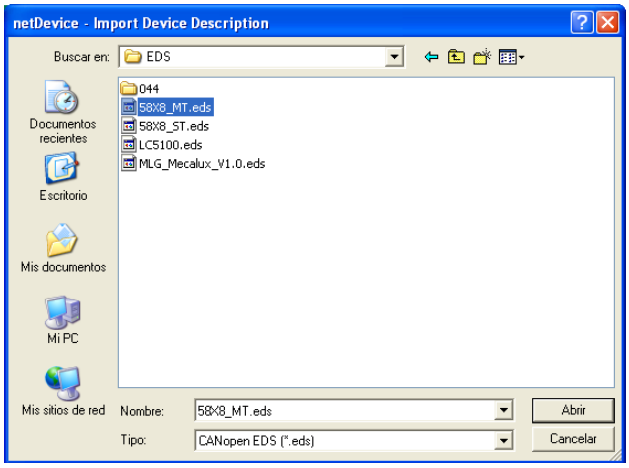
1.4.3.2.2 Instalación de EDS

Antes de continuar con cualquier configuración, bien sea de tarjeta o de esclavo, es necesario instalar los EDS correspondiente a cada uno de los dispositivos que

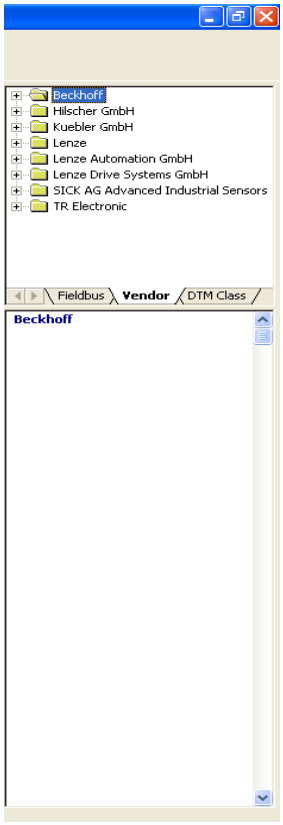
estarán conectados en el bus de campo. Para ello dentro del menú *Network*, seleccionar la opción *Import Device Descriptions….*



Aparecerá entonces el siguiente cuadro de diálogo en el que se seleccionará el path correspondiente para poder localizar el EDS adecuado. ,

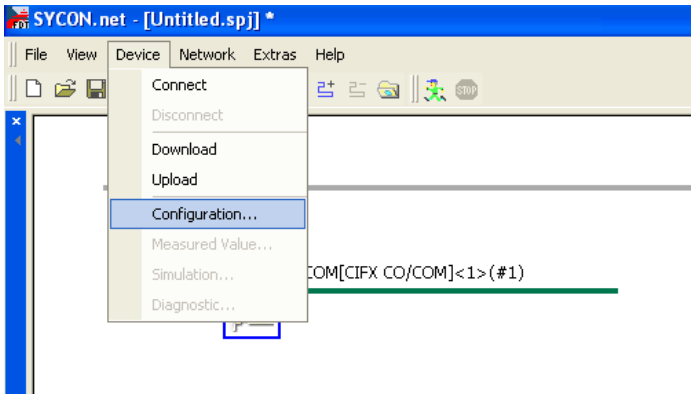


Una vez instalado, en el árbol de la parte derecha aparecerá dentro de la pestaña "Vendor" el fabricante con el dispositivo que ya ha sido insertado.



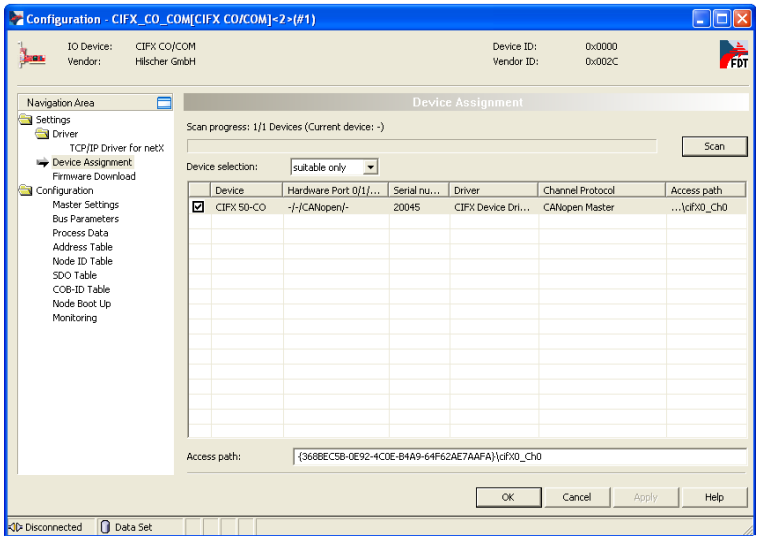
1.4.3.2.3 Configuración de la tarjeta

Se realiza la configuración de la tarjeta seleccionándola en la imagen y mediante el menú *Device* →*Configuration*:

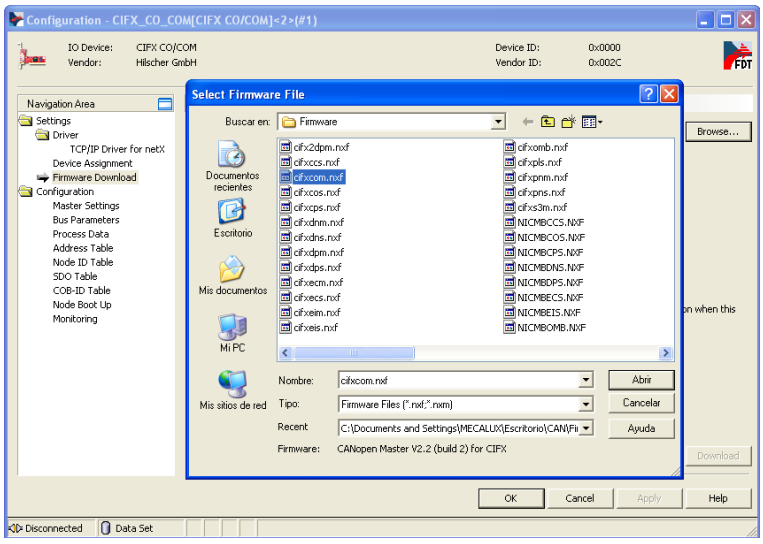


En la pantalla que aparece a continuación, es necesario realizar varios ajustes y comprobaciones referentes a la tarjeta CIFX_CO:

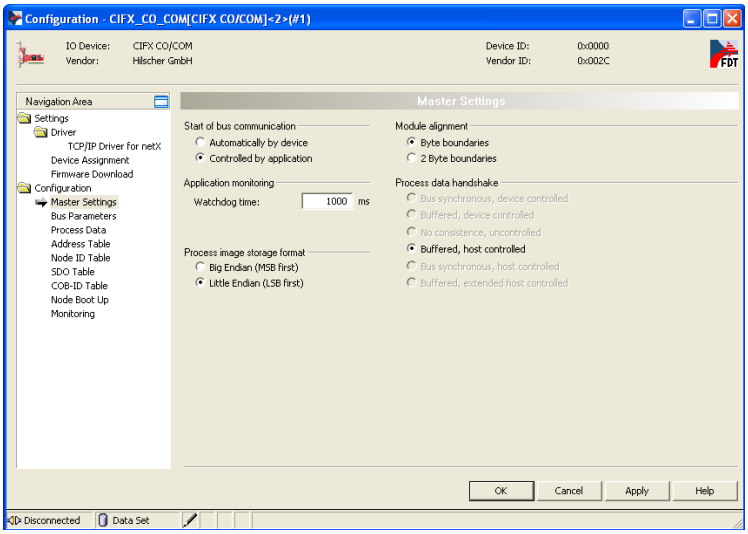
a. En el menú *Settings* → *Device Assignment*, es necesario verificar que la tarjeta utilizada aparece seleccionada y con el firmware adecuado (CANOPEN Master). Si no es así hay que realizar un "Scan" mediante el botón correspondiente y seleccionar dicha tarjeta.



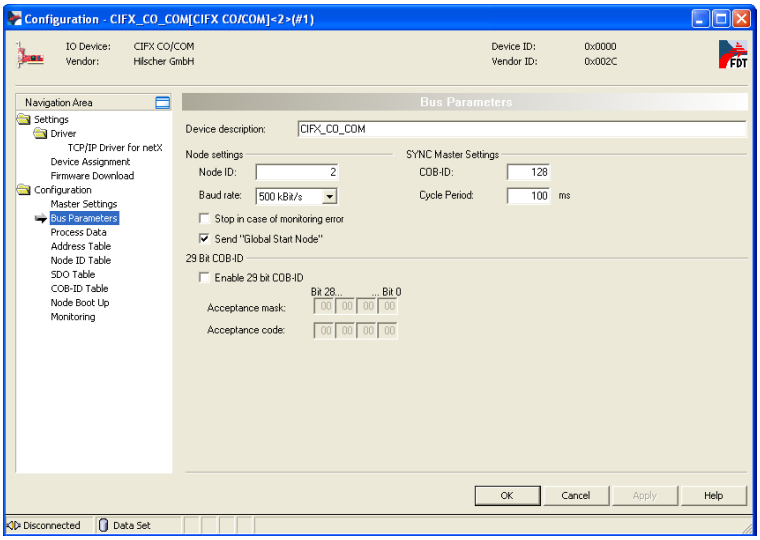
b. En el menú Setting → Firmware Download se da la posibilidad de poder instalarle un firmware si es necesario. Hay que tener en cuenta que a diferencia de las tarjetas utilizadas anteriormente, el firmware no reside en la propia tarjeta, si no en el PC en el que está alojada, por lo que si se produce un cambio de equipo, será necesario volver a actualizarla con el firmware correcto. El firmware que es necesario instalar para esta aplicación es "cifxcom.nxf" que corresponde al funcionamiento de la tarjeta como Master de CANOPEN.



c. En el menú Configuration → Master Settings se realizarán las configuraciones habituales marcando la tarjeta como "*Controlled by apllication*" y "*Buffered, host controlled*". Además hay que seleccionar la imagen de proceso como "*Little Endian(LSB first)*" que es el correspondiente al bus de campo CANOPEN.



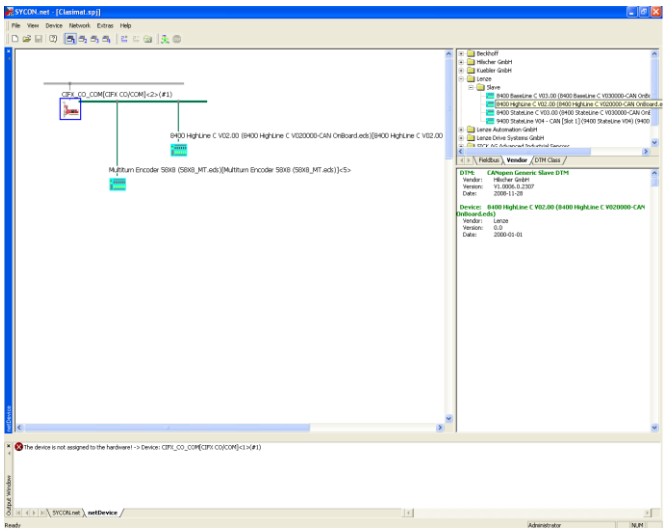
d. Dentro del menú Configuration → Bus Parameters, es necesario realizar la configuración del Node ID correspondiente (por defecto 2) y de la velocidad de transmisión que corresponda. Esta velocidad ha de ser configurada igual en todos y cada uno de los dispositivos que pertenezcan al bus de campo. Dentro de esta misma pantalla hay que marcar la opción "*Send "Global start Node"*".



Con estos ajustes iniciales, la tarjeta estaría totalmente configurada para poder comenzar a añadir los distintos elementos de bus.

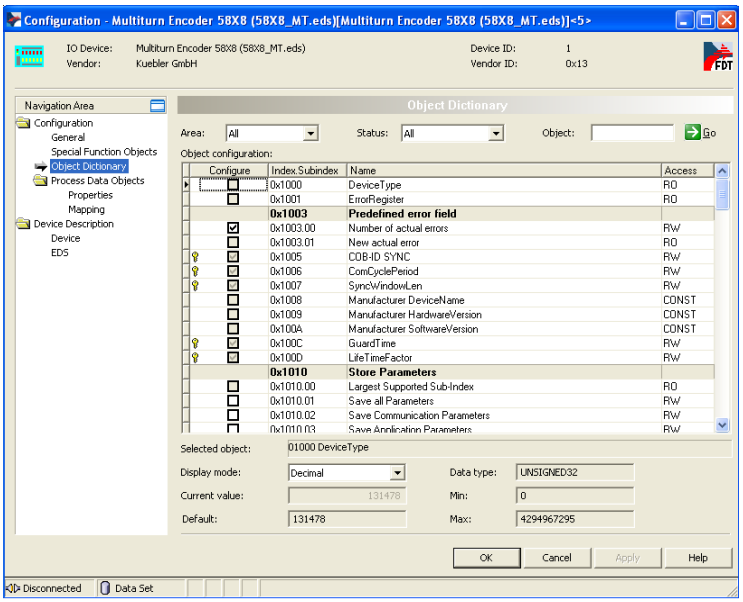
1.4.3.2.4 Añadir y configurar elementos de bus

Se selecciona el elemento hardware deseado en el árbol de la derecha dentro de la pestaña "*Vendor*" y se arrastra hasta la subred que se ha creado partiendo de la tarjeta insertada.

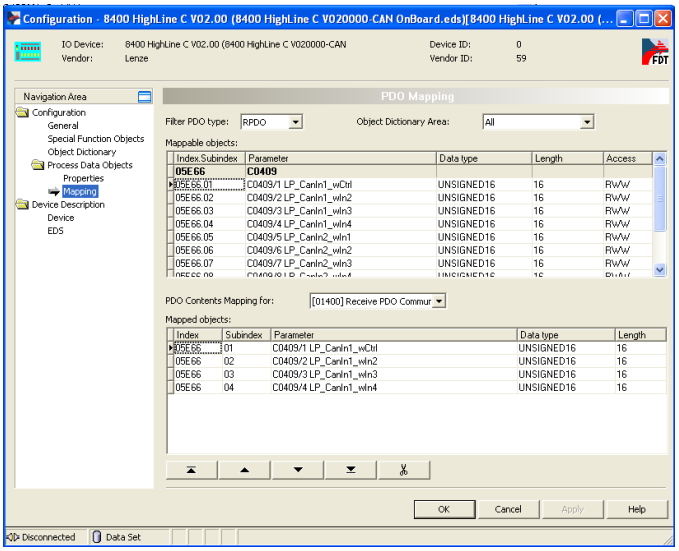


Una vez insertado el elemento, seleccionándolo dentro de la red, mediante el menú *Device* → *Configuration*, se pueden realizar una serie de ajustes que se detallan a continuación:

a. Menú *Configuration* → *Object Dictionary* se pueden activar los parámetros y valores que se deseen comunicar a través del bus. La selección a realizar en cada dispositivo hardware, vendrá indicada dentro de la documentación del componente del Sistema de Control Galileo.

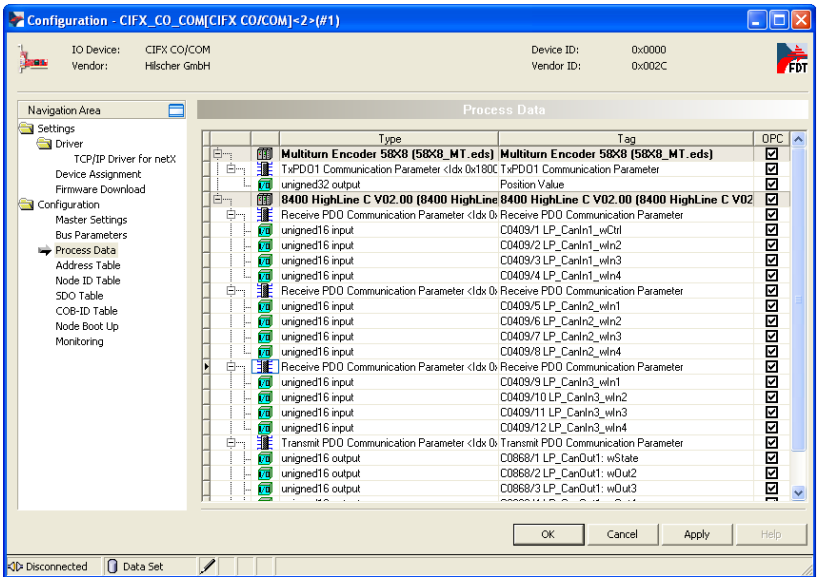


b. Menú *Configuration* → *Process Data Objects* → *Mapping*, se establece el contenido de cada una de las PDOs que sean utilizadas, bien sean de transmisión o de recepción. Esta configuración será descrita dentro de la documentación del elemento correspondiente en el Sistema de Control Galileo.

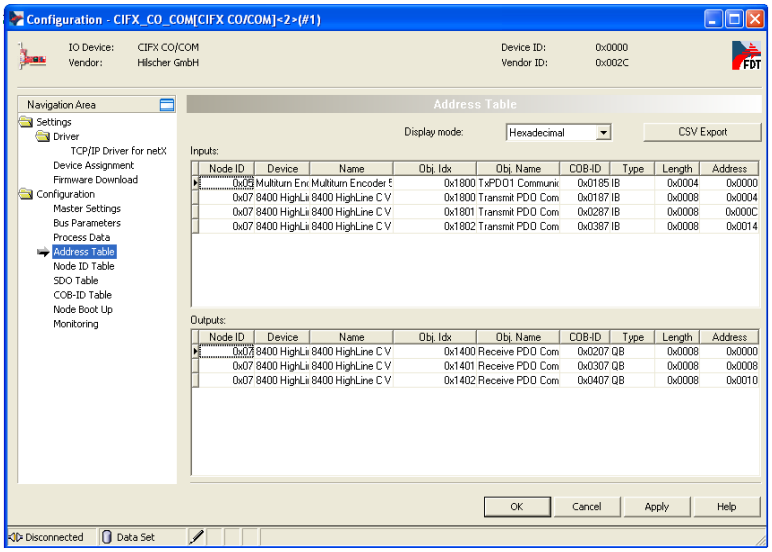


Una vez añadido y parametrizado cada dispositivo, es necesario volver a seleccionar la tarjeta de nuevo y mediante el menú ya utilizado *Device* → *Configuration* para terminar la configuración correcta:

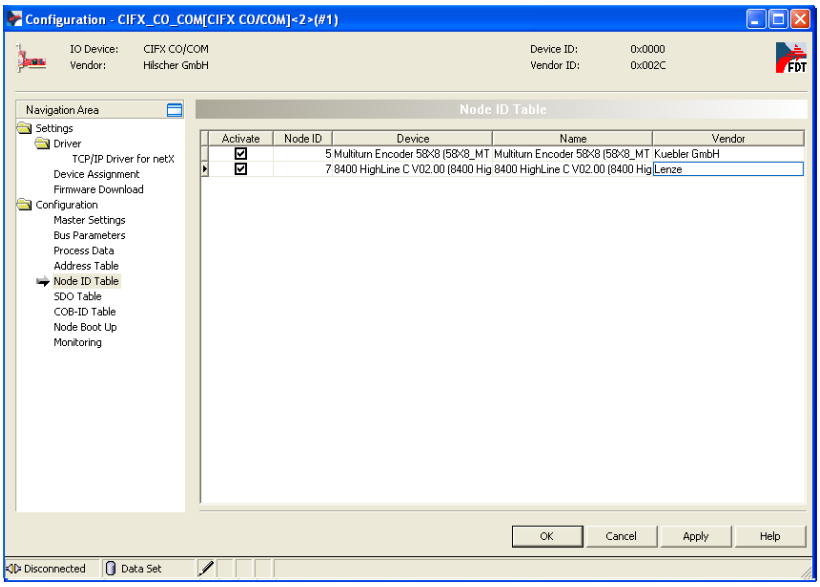
a. Menú *Configuration* → *Process Data* aparecerá un listado con todos y cada uno de los elementos que han sido configurados dentro de la red de bus de campo de la tarjeta, con el número de PDOs existente (tanto Transmit como Receive) y su configuración correspondiente en cuanto a tamaño, tipo, etc.



b. Menú *Configuration* → *Address Table* se muestra un mapeado completo de la periferia configurada, indicando tamaños, tipos, offset, etc.

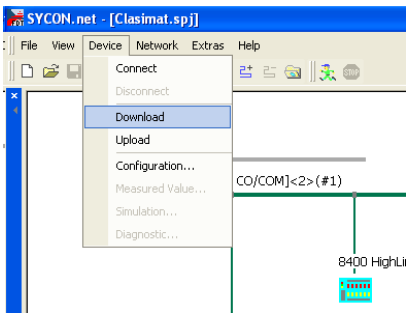


c. Menú *Configuration* → *Node ID Table* aparecen todos y cada uno de los elementos del bus de campo , teniendo que configurar aquí su Node ID, teniendo que ser este único dentro de la red del bus de campo. Es necesario también activar el elemento mediante el check "Activate".

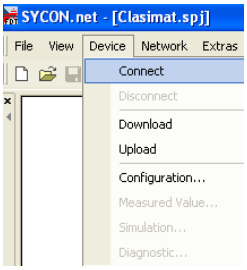


1.4.3.2.5 Arranque y diagnóstico del bus de campo

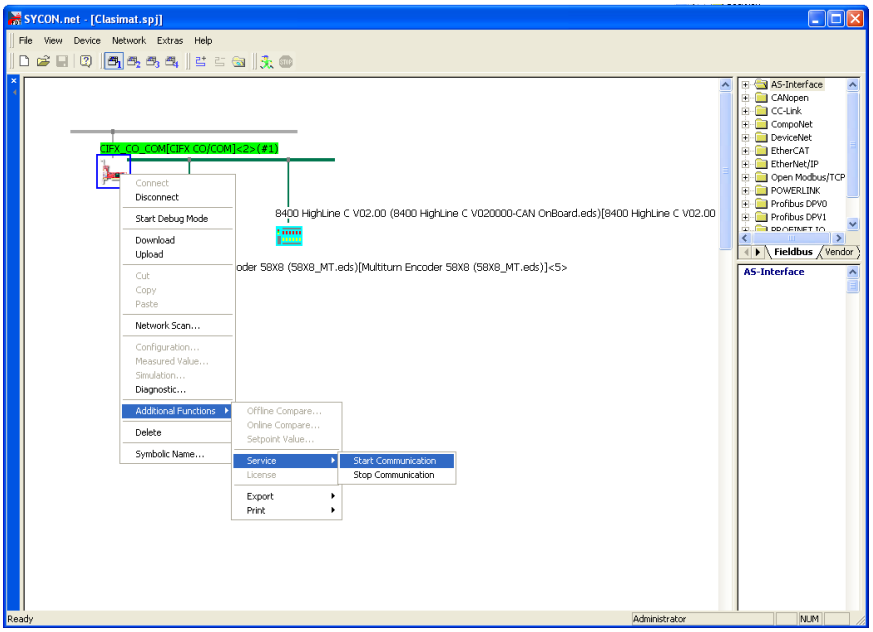
Una vez realizada la configuración completa, tanto de la tarjeta como de cada uno de los elementos que componen el bus de campo, es posible comenzar un diagnóstico antes de arrancar el programa del Sistema de Control Galileo. Para ello antes de nada hay que realizar una descarga de dicha configuración a la tarjeta mediante la opción Device → *Download*.



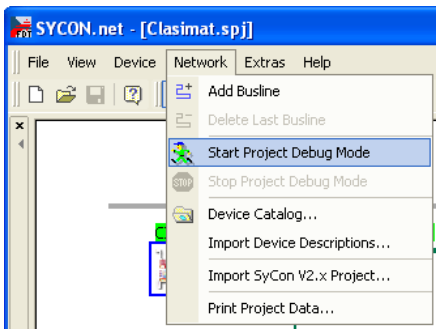
1. Mediante la opción del menú *Device* → *Connect*, se establece conexión con la tarjeta.



2. Pinchando con el botón derecho sobre el icono de la tarjeta, aparece el menú *Adicional Functions* → *Service* → *Start Comunication*, para que la tarjeta inicie la comunicación con los dispositivos.



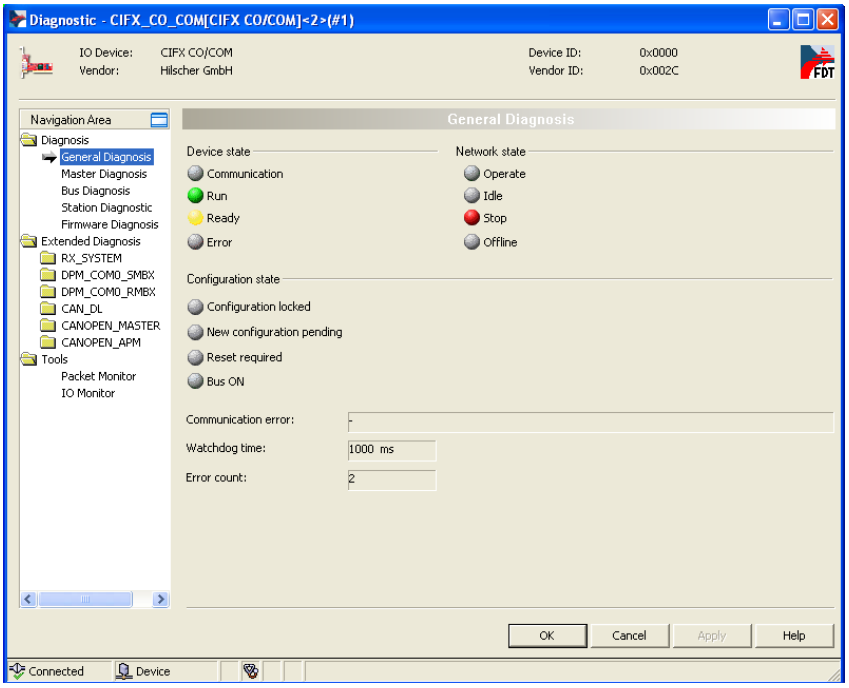
3. Una vez que la comunicación ya se ha establecido sin errores, se puede diagnosticar el estado de la red mediante la opción del menú *Network* → *Start Project Debug Mode*, que muestra mediante colores verde (OK) y rojo (error) el estado de cada uno de los dispositivos.



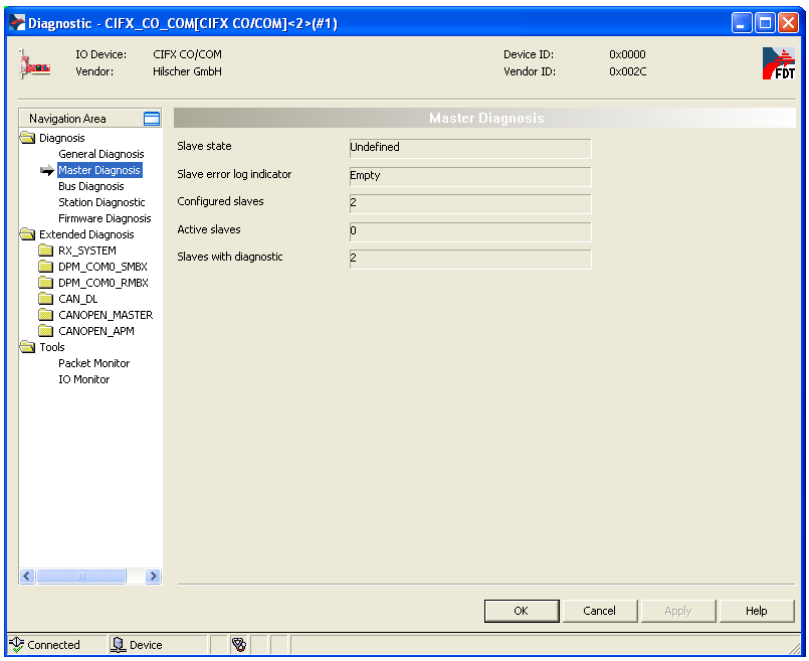
4. Dentro del modo debug, es posible realizar un diagnóstico más profundo del estado de las comunicaciones, seleccionando la tarjeta en la configuración y dentro del menú *Device* → *Diagnostic.*

Aparece la siguiente ventana con distintas opciones de diagnóstico:

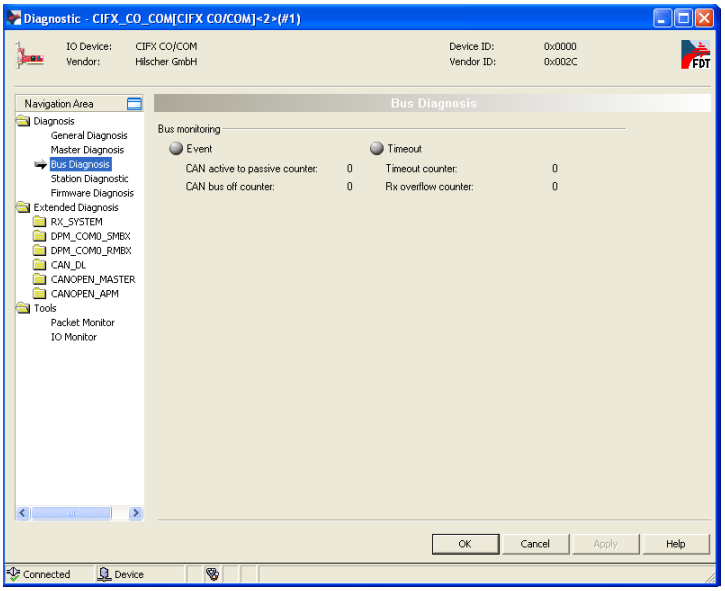
4.1 *Diagnosis* → *General Diagnosis*.- Permite ver el estado de la tarjeta, de la red y si el bus se encuentra o no operativo. Muestra además un mensaje de error de comunicación si existiese.



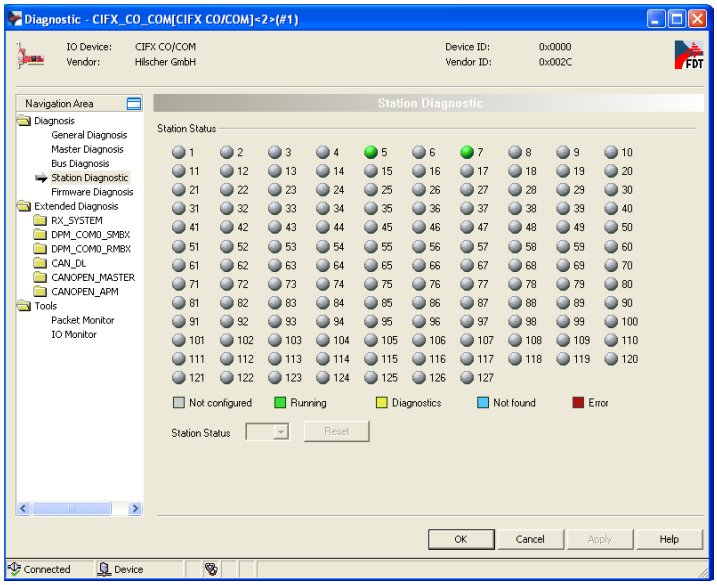
4.2 *Diagnosis* → *Master Diagnosis*.- Muestra si existen errores en algún esclavo e indica el número de esclavos configurados y cuantos de ellos se encuentran activos.



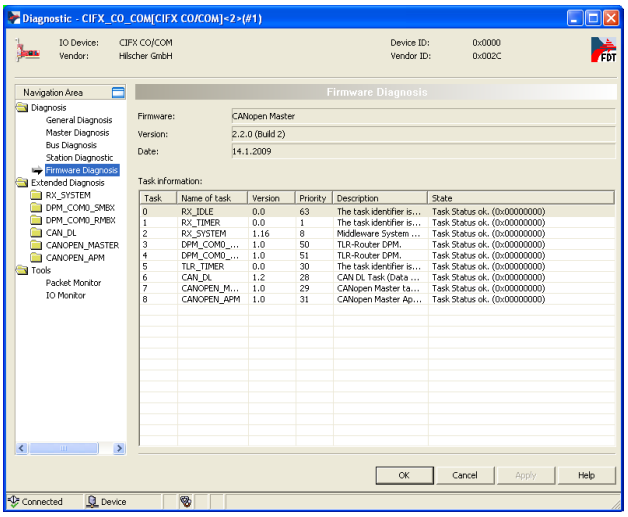
4.3 *Diagnosis* → *Bus Diagnosis*.- Indica si se han producido errores de transmisión o de timeout en el bus.



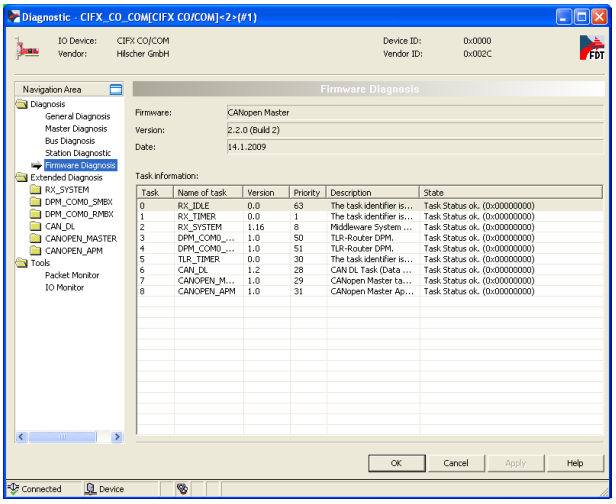
4.4 *Diagnosis* → *Station Diagnosis*.- Muestra un listado con el Node ID de los esclavos, y mediante un LED indica si están comunicando, en modo diagnóstico, perdidos o en error.



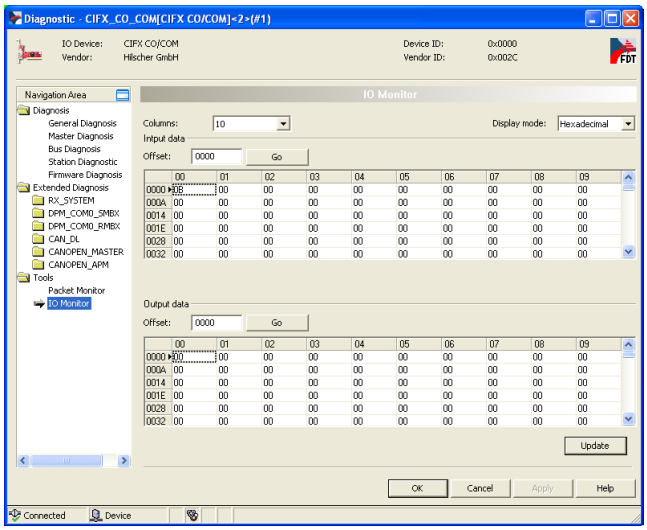
4.5 *Diagnosis* → *Firmware Diagnosis*.- Muestra los datos de versión y revisión del firmware instalado, así como las tareas que se están ejecutando junto con su estado (Task State).



4.6 *Tools* → *Packet Monitor*.- Es posible configurar el envío o recepción de datos de la periferia. Esta opción es idéntica a la ya existente con las anteriores versiones de Sycon, con los datos de comunicación específicos de CANOPEN.



4.7 *Tools* → *IO Monitor*.- Muestra una imagen tanto de la periferia de entradas como de la de salidas actual de la tarjeta.



Una vez configurada correctamente la tarjeta y los elementos del bus de campo CANOPEN y verificado su correcto funcionamiento y comunicación, ya es posible realizar el manejo desde la aplicación de control generada para tal fin en el Sistema de Control Galileo.

1.5 Bus de campo MODBUS

1.5.1 Introducción

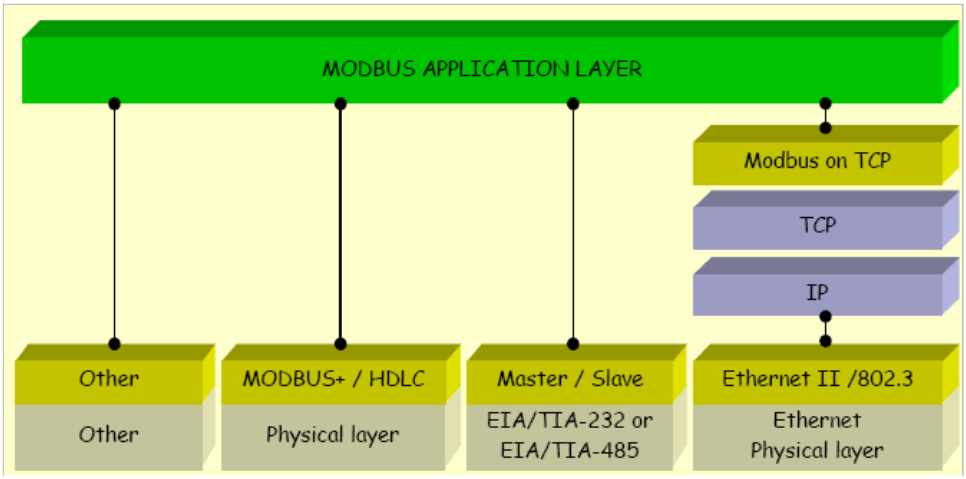
MODBUS es una protocolo de mensajería de la capa de aplicación, situado en el nivel 7 del modelo OSI, que proporciona un modelo cliente/servidor para comunicar dispositivos conectados en diferentes tipos de buses o redes.

Es estándar de facto desde 1979, y actualmente MODBUS continúa permitiendo la comunicaciones a millones de dispositivos. Hoy por hoy, la estructura sencilla de este protocolo sigue creciendo. La comunidad de Internet puede acceder al protocolo MODBUS a través del puerto 502 de TCP.

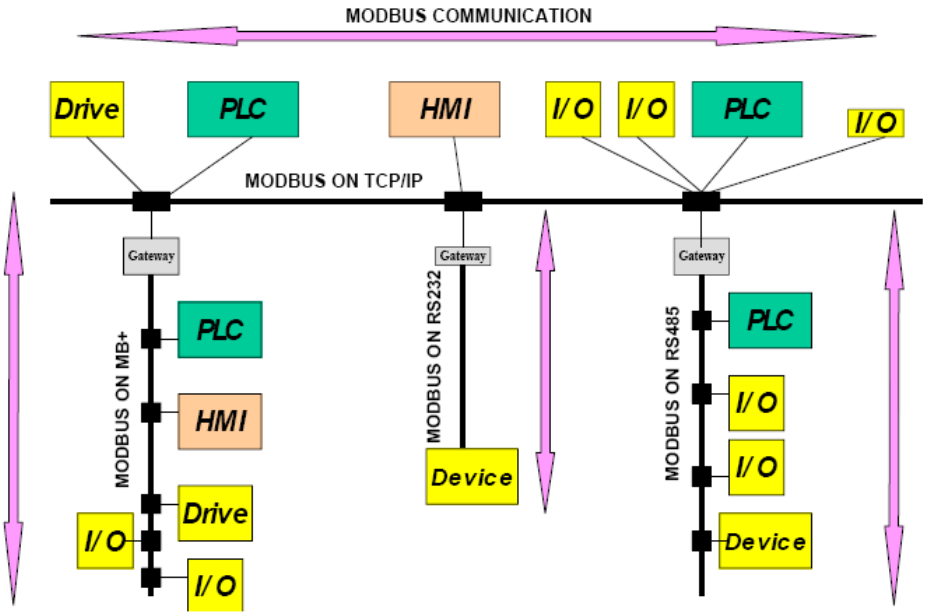
MODBUS es un protocolo de pregunta/respuesta que ofrece servicios espaciados por códigos de función. Estos códigos son conocidos como PDUs.

Actualmente, MODBUS esta implementado utilizado:

- TCP/IP sobre Ethernet
- Transmisión serie asíncrona sobre una variedad de medios (EIA/TIA-232-E, EIA-422, EIA/TIA-485-A, fibra óptica, radio, etc)
- MODBUS PLUS, una red de alta velocidad basada en el paso de testigo



El protocolo MODBUS permite una comunicación sencilla dentro de cualquier topología de red.



Cualquier tipo de dispositivo (PLC, HMI, panel de control, etc.) puede usar este protocolo para inicializar una operación remota.

La misma comunicación se puede realizar mediante una línea serie o sobre una red Ethernet TCP/IP. La utilización de GateWays posibilita la comunicación entre varios tipos de buses utilizando el protocolo MODBUS.

1.5.2 Estructura de la red

1.5.2.1 Medio Físico

Como medio físico de conexión, las opciones posibles son:

- Un bus half-duplex (RS-485 o fibra optica)
- Un bus full-duplex (RS-422, lazo de corriente 0-20 mA o fibra óptica)

En cualquier caso, se trata de comunicaciones asíncronas, con velocidades de transmisión que van desde los 75 Bits/s hasta los 19200 Bits/s. Existen así mismo, limitaciones en la distancia máxima entre estaciones, y dependiendo del nivel físico utilizado, pueden alcanzar los 1200 metros sin la utilización de repetidores.

1.5.2.2 Acceso al medio

La red se estructura con una topología maestro-esclavo, con acceso al medio controlado por el maestro. El número máximo de estaciones es de 64, de las que una será la estación maestra.

Se pueden distinguir dos tipos de mensajes: los mensajes punto a punto, que se componen de una petición de maestro y una respuesta desde el esclavo, los mensajes difundidos, que son enviados desde el maestro a todos los esclavos y no tienen respuesta. Este tipo de mensajes se utiliza para transmitir datos comunes, como parámetros de configuración, etc.

La manera en la que se controla el acceso al medio es mediante la técnica de paso de testigo. En esta técnica, solo la estación en posesión del testigo (una trama especial) tiene el derecho de utilizar el bus.

1.5.3 Protocolo

Existen dos maneras de codificar las tramas de datos en el protocolo ModBus. Por un lado se puede realizar en modo ASCII o en modo binario, según el estándar RTU. Realmente, ambas codificaciones son idénticas, y la única diferencia está en que la trama ASCII esta flanqueada por una serie de caracteres especiales:

- El carácter ‘:’ (3Ah) al inicio
- Los caracteres CR (0Dh) y LF (0Ah) al final

Otra diferencia añadida está en la manera en que se calcula el CRC que se incorpora en la trama. En la trama ASCII se utiliza una suma en modulo 16, mientras que en la trama binaria un formula polinómica. A continuación se muestra

el formato general de ambos tipos de trama y una descripción de sus campos principales:

3Ah	Nº esclavo (00 – 3Fh)	Código de operación	Subfunciones, datos	LHR(16) HL	CR (0Dh)	LF (0AH)

Trama en formato ASCII

Nº esclavo (00 – 3Fh)	Código de operación	Subfunciones, datos	LHR(16) HL

Trama en formato binario

- *Número de esclavo (1 byte):* Permite direccionar un máximo de 63 esclavos con direcciones que van del 01_H hasta $3F_H$. El número 00_H se reserva para los mensajes difundidos.

- *Código de operación o función (1 byte):* Los códigos de funciones permiten especificar los comandos, órdenes o datos que se quieren transmitir al esclavo. Existen diversos códigos de orden, pero se pueden clasificar en los siguientes dos tipos:

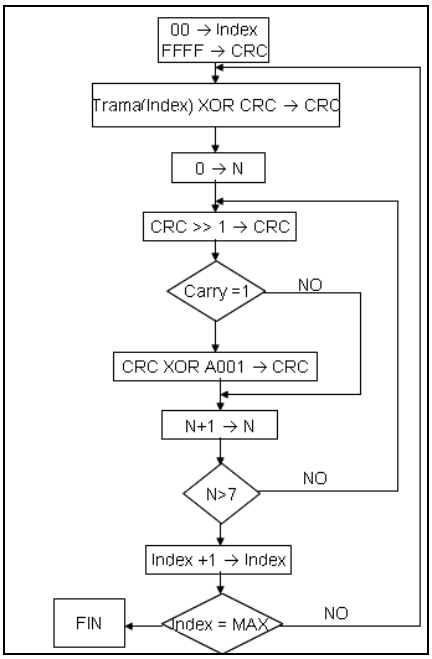
    - Ordenes de lectura/escritura de datos en los registros o en la memoria del esclavo.
    - Ordenes de control del esclavo y el propio sistema de comunicaciones (RUN/STOP, carga y descarga de programas, verificación de contadores de intercambio, etc.)

En la siguiente tabla se muestran los códigos de función principales y los datos que necesitan:

Código subfunción		Datos subfunción		Tarea
SF0	SF1	D0	D1	
00h	00h	00h	00h	Paro del esclavo sin inicializar
00h	01h	00h	00h	Marcha del esclavo sin inicializar
00h	02h	00h	00h	Marcha e inicialización del esclavo
00h	03h	00h	XXh	Lectura de la secuencia XXh de programa de usuario en el esclavo
00h	04h	YYh	XXh	Carga de una secuencia de programa de usuario en el esclavo **Petición:** YY = secuencia a cargar, XX= próxima secuencia **Respuesta:** XX= código error, YY= 00

- *Campo de subfunciones/datos (n bytes)*: En este campo se pueden transmitir, por un lado, los datos que necesitan las funciones mostradas en la tabla anterior, o por otro, codigos de funciones en el caso de órdenes de control.

- *Palabra de control de errores (2 bytes)*: En código ASCII, esta palabra es simplemente la suma de comprobación ('*checksum*') del mensaje en módulo 16 expresado en ASCII. En el caso de codificación RTU el CRC se calcula con una fórmula polinómica según el algoritmo mostrado en la siguiente figura:



1.5.3.1 Descripción de las funciones del protocolo

1.5.3.1.1 Función 0

La función 0 se utiliza para transmitir a los esclavos ordenes de control como marcha o paro. Como se dijo antes, es el campo de datos o subfunciones en el que se cargan los códigos de subfunción en los mensajes de control. En las siguientes figuras se pueden ver las tramas de petición del maestro, así como una descripción de las distintas subfunciones posibles. En caso de las órdenes de marcha y paro, el campo de «información» de la trama está vacío y, por tanto, el mensaje se compone simplemente de 6 `bytes` de función más 2 `bytes` de CRC. La respuesta del esclavo a estas órdenes es un mensaje idéntico al enviado por el maestro. Cabe señalar, además, que después de un paro el autómata sólo acepta ejecutar subfunciones de la función 00_H.

Función	Código	Tarea
0	00 h	Control de las estaciones esclavas
1	01 h	Lectura de n bits de salidas o internos
2	02 h	Lectura de n bits de entradas
3	03 h	Lectura de n palabras de salidas o internos
4	04 h	Lectura de n palabas de entradas
5	05 h	Escritura de un bit
6	06 h	Escritura de una palabra
7	07 h	Lectura rápida de 8 bits

8	08 h	Control de contadores de diagnostico número 1 a 8
9	09 h	No utilizado
10	0A h	No utilizado
11	0B h	Control de contador de diagnostico número 9
12	0C h	No utilizado
13	0D h	No utilizado
14	0E h	No utilizado
15	0F h	Escritura de n bits
16	10 H	Escritura de n palabras

Formato petición

Nº Esclavo (00 – 3Fh)	00h	Código ó subfunción	Datos subfunción	Información	CRC	
					H	L

1.5.3.1.2 Funciones 1 y 2

Lectura de bits del autómata. La trama es la indicada en la figura. La forma de direccionamiento de los bits es a base de dar la dirección de la palabra que los contiene y luego la posición del `bit`. Obsérvese también que la respuesta es dada siempre en `bytes` completos.

Formato petición

Nº Esclavo (00 – 3Fh)	01h/02h	Dirección 1er bit		Nº de bits		CRC	
		PP	PB	NN	NN	H	L

PPP: dirección de la palabra, B: posición del bit dentro de la palabra

Formato respuesta

Nº Esclavo (00 – 3Fh)	01h/02h	Nº bytes leidos		Primer bytes			Otros bytes	CRC	
		NN	NN	B1	…	B7	Max 256	H	L

1.5.3.1.3 Funciones 3 y 4

Lectura de palabras del autómata. La trama es la indicada en la figura. Obsérvese que la petición indica el número de palabras a leer, mientras que en la respuesta se indica el número de `bytes` leídos.

Formato petición

Nº Esclavo (00 – 3Fh)	03h/04h	Dirección 1era palabra		Nº de palabras		CRC	
		PP	PP	NN	NN	H	L

PPPP: dirección de la pabra

Formato respuesta

Nº Esclavo (00 – 3Fh)	03h/04h	Nº de bytes leidos		Primera palabra			Otras palabras max 128			CRC	
		NN	NN	B1	…	B7	HL	…	HL	H	L

1.5.3.1.4 Función 5

Escritura de un `bit`. La trama es la indicada en la figura. El direccionamiento del `bit` se efectúa tal como se ha indicado para las funciones 1 y 2.

Formato petición

Nº Esclavo (00 – 3Fh)	05h	Dirección bit		Valor 00h→0 FFh→1	00h	CRC	
		PP	PB			H	L

PPP: dirección de la palabra; B: dirección del bit dentro de la palabra

Formato respuesta

Nº Esclavo (00 – 3Fh)	05h	Dirección bit		Valor 00h→0 FFh→1	00h	CRC	
		PP	PB			H	L

1.5.3.1.5 Función 6

Escritura de una palabra. La trama es la que se muestra a continuación:

Formato petición

Nº Esclavo (00 – 3Fh)	06h	Dirección palabra		Valor palabra		CRC	
		PP	PP	DD	DD	H	L

Formato respuesta

Nº Esclavo (00 – 3Fh)	06h	Dirección palabra		Valor palabra		CRC	
		PP	PP	DD	DD	H	L

1.5.3.1.6 Función 7

Petición de lectura rápida de un `byte`. Obsérvese que la petición no tiene campo de dirección, esto es debido a que el `byte` legible por esta función es fijo en cada esclavo y viene fijado en su configuración.

Formato petición

Nº Esclavo (00 – 3Fh)	07h	CRC	
		H	L

Formato respuesta

Nº Esclavo (00 – 3Fh)	07h	Valor leido	CRC	
			H	L

1.5.3.1.7 Función 8

Petición del contenido y control de los 8 primeros contadores de diagnóstico de un esclavo.

Formato petición

Nº Esclavo (00 – 3Fh)	08h	Código subfunción	Dato subfunción	CRC

		SF0	SF1	D0	D1	H	L

Formato respuesta

Nº Esclavo (00 – 3Fh)	08h	Código subfunción		Valor contador		CRC	
		SF0	SF1	H	L	H	L

Subfunción			Datos		Tareas
Nº	Código sub.		D0	D1	
0	00h	00h	XYh	ZTh	El esclavo envía el eco XYZT de petición como test
3	00h	03h	ZZh	00h	Modifica el carácter de fin de trama en modo ASCII por ZZh
10	00h	0Ah	00h	00h	Puesta a cero de los contadores
11	00h	0Bh	00h	00h	Lectura del contador 1 (Bus Message Count): número de mensajes que el dispositivo remoto ha detectado desde el último reinicio, borrado de contadores o encendido.
12	00h	0Ch	00h	00h	Lectura del contador 2 (Bus Comunication Error Count): número de errores de CRC que el dispositivo remoto ha detectado desde el último reinicio, borrado de contadores o encendido.
13	00h	0Dh	00h	00h	Lectura del contador 3 (Bus Exception Error Count): número de respuestas de excepción ModBus enviadas por el esclavo desde el último reinicio, borrado de contadores o encendido.
14	00h	0Eh	00h	00h	Lectura del contador 4 (Slave Message Count): número de mensajes dirigidos al esclavo (o multidifundidos) que éste a procesado desde el último reinicio, borrado de contadores o encendido.
15	00h	0Fh	00h	00h	Lectura del contador 5 (Slave No Response Count): número de mensajes dirigidos al esclavo que no han sido respondidos (normal o respuesta de excepción) desde el último reinicio, borrado de contadores o encendido.
16	00h	10h	00h	00h	Lectura del contador 6 (Slave NAK Count): número de mensajes dirigidos al esclavo para los que éste ha respondido con una respuesta negativa NAK.
17	00h	11h	00h	00h	Lectura del contador 7 (Slave Busy Count): número de mensajes dirigidos al esclavo para los que éste ha devuelto un mensaje de excepción "Slave Busy".
18	00h	12H	00h	00h	Lectura del contado 8 (Bus Carácter Overrun Count): número de mensajes dirigidos al esclavo que no han podido ser procesados por producirse una condicion "overrun". Este error se produce cuando llegan datos al puerto más rápido de lo que pueden ser almacenados o por un error de hardware.

1.5.3.1.8 Función 11

La petición del contenido del contador de diagnóstico número 9, no se realiza por la función 8, sino por la función 11.

Formato petición

Nº Esclavo (00 – 3Fh)	0Bh	CRC	
		H	L

Formato respuesta

Nº Esclavo (00 – 3Fh)	0Bh			Valor contador		CRC	
		00	00	H	L	H	L

1.5.3.1.9 Función 15

Escritura de bits del autómata. La forma de direccionamiento es análoga a la indicada para las funciones 1 y 2.

Formato petición

Nº Esclavo (00 – 3Fh)	0Fh	Dirección del primer bit		Nº de bits		Nº de bytes	Valor de los bits	CRC	
		PP	PB	NN	NN	M	8xM Valores	H	L

Formato respuesta

Nº Esclavo (00 – 3Fh)	0Fh	Dirección del primer bit		Nº de bits		CRC	
		PP	PB	NN	NN	H	L

1.5.3.1.10 Función 16

Escritura de palabras del autómata.

Formato petición

Nº Esclavo (00 – 3Fh)	10h	Dirección de la 1ra palabra		Nº de palabras		Nº de bytes	Valor de las palabras			CRC	
		PP	PP	NN	NN	M	HL	..	HL	H	L

Formato respuesta

Nº Esclavo (00 – 3Fh)	10h	Dirección de la 1ra palabra		Nº de palabras		CRC	
		PP	PP	NN	NN	H	L

1.5.3.1.11 Mensajes de error

Puede ocurrir que un mensaje se interrumpa antes de terminar. Cada esclavo interpreta que el menaje ha terminado si transcurre un tiempo de silencio equivalente a 3.5 caracteres. Después de este tiempo el esclavo considera que el carácter siguiente es el campo de dirección de esclavo de un nuevo mensaje. Cuando un esclavo recibe una trama incompleta o errónea desde el punto de vista lógico, envía un mensaje de error como respuesta, excepto en el caso de mensajes de difusión.

Nº Esclavo (00 – 3Fh)	Código función	Código error	CRC	Código Función = Código función recibido + 80H
				Código Error

		PP	PP	NN	NN	01 → Código de Función rróneo 02 → Dirección incorrecta 03 → Datos Incorrectos 04 → Autómata ocupado

Si la estación maestra no recibe respuesta de un esclavo durante un tiempo superior a un límite establecido, declara el esclavo fuera de servicio, a pesar de que al cabo de un cierto número de ciclos hace nuevos intentos de conexión.

1.5.4 MODBUS® TCP/IP

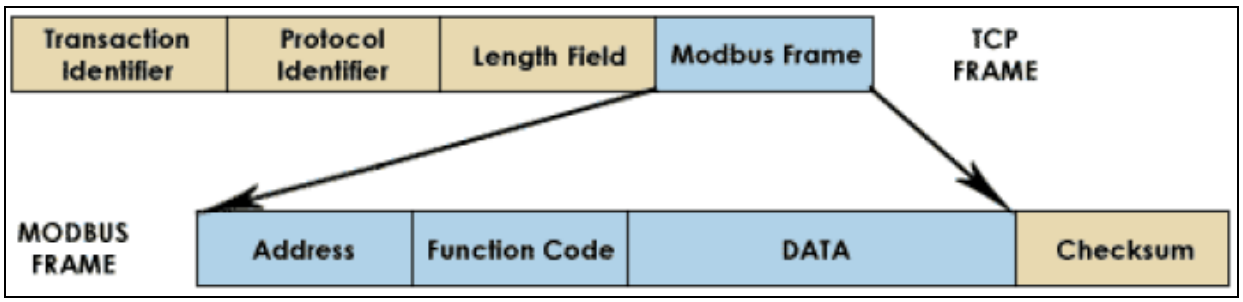
MODBUS® TCP/IP es una variante o extensión del protocolo Modbus que permite utilizarlo sobre la capa de transporte TCP/IP. De este modo, Modbus-TCP se puede utilizar en Internet, de hecho, este fue uno de los objetivos que motivó su desarrollo (la especificación del protocolo se ha remitido a la IETF=Internet Engineering Task Force). En la práctica, un dispositivo instalado en Europa podría ser direccionado desde EEUU o cualquier otra parte del mundo. Las ventajas para los instaladores o empresas de automatización son innumerables:

- Realizar reparaciones o mantenimiento remoto desde la oficina utilizando un PC, reduciendo así los costes y mejorando el servicio al cliente.
- El ingeniero de mantenimiento puede entrar al sistema de control de la planta desde su casa, evitando desplazamientos.
- Permite realizar la gestión de sistemas distribuidos geográficamente mediante el empleo de las tecnologías de Internet/Intranet actualmente disponibles.

MODBUS® TCP/IP se ha convertido en un estándar industrial *de facto* debido a su simplicidad, bajo coste, necesidades mínimas en cuanto a componentes de hardware, y sobre todo a que se trata de un protocolo abierto. En la actualidad hay cientos de dispositivos MODBUS® TCP/IP disponibles en el mercado. Se emplea para intercambiar información entre dispositivos, así como monitorizarlos y gestionarlos. También se emplea para la gestión de entradas/salidas distribuidas, siendo el protocolo más popular entre los fabricantes de este tipo de componentes. La combinación de una red física versátil y escalable como Ethernet con el estándar universal de interredes TCP/IP y una representación de datos independiente de fabricante, como MODBUS®, proporciona una red abierta y accesible para el intercambio de datos de proceso.

1.5.4.1 El protocolo Modbus TCP

Modbus/TCP simplemente encapsula una trama Modbus en un segmento TCP. TCP proporciona un servicio orientado a conexión fiable, lo que significa que toda consulta espera una respuesta.



Esta técnica de consulta/respuesta encaja perfectamente con la naturaleza Maestro/Esclavo de Modbus, añadido a la ventaja del determinismo que las redes Ethernet conmutadas ofrecen a los usuarios en la industria. El empleo del protocolo abierto Modbus con TCP proporciona una solución para la gestión desde unos pocos a decenas de miles de nodos.

1.5.4.2 Prestaciones de un sistema MODBUS TCP/IP

Las prestaciones dependen básicamente de la red y el hardware. Si se usa MODBUS® TCP/IP sobre Internet, las prestaciones serán las correspondientes a tiempos de respuesta en Internet, que no siempre serán las deseables para un sistema de control. Sin embargo pueden ser suficientes para la comunicación destinada a depuración y mantenimiento, evitando así desplazamientos al lugar de la instalación.

Si disponemos de una Intranet de altas prestaciones con conmutadores Ethernet de alta velocidad, la situación es totalmente diferente.

En teoría, MODBUS® TCP/IP, transporta datos hasta 250/(250+70+70) o alrededor de un 60% de eficiencia cuando se trasfieren registros en bloque, y puesto que 10 Base T proporciona unos 1.25 Mbps de datos, la velocidad de transferencia de información útil será:

1.25M / 2 * 60% = 360000 registros por Segundo

En 100BaseT la velocidad es 10 veces mayor. Esto suponiendo que se están empleando dispositivos que pueden dar servicio en la red Ehernet aprovechando todo el ancho de banda disponible.

En los ensayos prácticos realizados por by Schneider Automation utilizando un PLC Ethernet Momentum™ con entradas/salidas Ethernet, demostró que se podían escanear hasta 4000 bloques I/O por segundo, cada uno con hasta 16 I/O analógicas de 12-bits o 32 I/O digitales (se pueden actualizar 4 bases por milisegundo). Aunque estos resultados están por debajo del límite teórico calculado anteriormente, pero debemos recordar que el dispositivo se probó con una CPU de baja velocidad (80186 a 50MHz con 3 MIPS).

Además, el abaratamiento de los ordenadores personales y el desarrollo de redes Ethernet cada vez más rápidas, permite elevar las velocidades de funcionamiento, a diferencia de otros buses que están inherentemente limitados una sola velocidad.

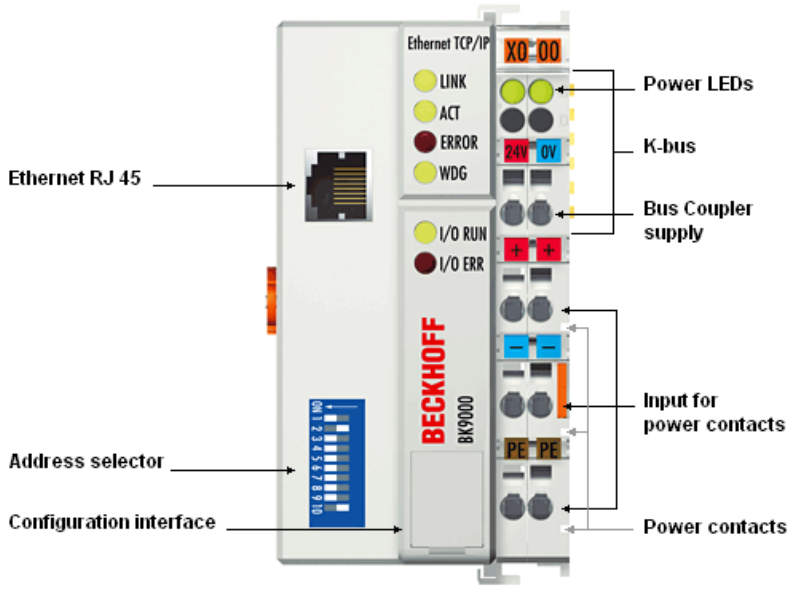
1.5.5 ModBus en el Sistema de Control Galileo

Galileo permite la utilización del protocolo MODBUS para el acceso a los módulos **Beckhoff BK9000**. Se trata de unas cabeceras a las que se les conectan diversos módulos de E/S y que permite su acceso a través del protocolo **ModBus-TCP**. A continuación se muestran los pasos para la configuración de estos dispositivos.

1.5.5.1 Configuración de los módulos BK9000

1.5.5.1.1 Consideraciones previas

Existen distintos métodos para la asignación de una IP a los módulos BK9000 de Beckhoff. Algunos de estos métodos son manuales, actuando sobre los interruptores DIP de configuración del módulo, y otros automáticos, mediante servidores DHCP o Boota, o mediante software especifico de configuración. Existen también procedimientos por los que se devuelve al módulo al estado por defecto, lo que puede servir para comenzar de nuevo la configuración en caso de errores. Comenzaremos explicando primero estos métodos de restauración del dispositivo a la configuración por defecto.



1.5.5.1.2 Resetear el dispositivo a la configuración por defecto

En caso de necesidad, el dispositivo puede ser configurado con unos parámetros por defecto proporcionados por el fabricante (parámetros referidos a velocidades de transmisión). Los pasos a seguir para volver a la configuración por defecto es:

- Con el módulo BK9000 apagado, conectar ÚNICAMENTE el módulo terminal de bus (BK9010).
- Colocar todos los interruptores DIP a la posición ON.
- Encender el modulo BK9000.

- Una vez se haya cargado la configuración por defecto, los LEDs de *Error, I/O Run* y *I/O Error*, parpadearán alternativamente.
- Apagar el módulo BK9000 y continuar con el funcionamiento normal

1.5.5.1.3 Establecimiento de los parámetros de la red Ethernet

Es posible modificar los parámetros referidos a velocidad de transmisión del módulo BK9000 de Beckhoff. Para ellos es necesario seguir estos pasos:

- Con el módulo BK9000 apagado, conectar ÚNICAMENTE el módulo terminal de bus (BK9010).
- Colocar los interruptores DIP 1 a 9 a la posición ON, y el 10 a OFF.
- Encender el modulo BK9000.
- El modulo mantendrá los de *Error, I/O Run* y *I/O Error* encendidos.
- Colocar los interruptores 1 a 3, en la posición deseada, según esta tabla:

Interruptor	Parámetro	Selección	Posición del DIP	Comentario
1	Velocidad de transmisión	10 Mbits	Off	
		100 MBits	On	Por defecto
2	Auto selección de velocidad	Activada	Off	
		Desactivada	On	Por defecto
3	Tipo de transmisión	Half duplex	Off	
		Full Duplex	On	Por defecto

1.5.5.1.4 Asignación de IP de manera manual

Es posible la asignación de manera manual de una IP a los módulos BK9000. Este suele ser el mecanismo más sencillo de hacerlo, aunque tiene muchas limitaciones. En caso de hacerlo de esta manera, solo se puede indicar una IP de la red 172.16.17, es decir una clase C. Los pasos a seguir son:

- Con el modulo BK9000 apagado, colocar los DIP 9 y 10 a OFF.
- Codificar en los DIP 1-8 la dirección que se quiere asignar (0 – 255)
- Encender el módulo BK9000

1.5.5.1.5 Asignación de IP de manera automática con un servidor Boota

Es posible que los módulos obtengan una dirección de manera automática a través de un servidor BootP, de manera similar a como lo hacen los módulo FL IL 24 BK que se verán en el apartado dedicado Bluetooth. Para ello es necesario que los módulos estén conectados a una red que disponga de este servidor. Para más datos acerca de la configuración de un servidor BootP, consultar el apartado dedicado a BlueTooth. Los pasos a seguir para configurar el módulo son los siguientes:

- Con el modulo BK9000 apagado, colocar el DIP 9 a ON y el 10 a OFF.

- Encender el módulo BK9000.

Dependiendo de la posición de los DIP 1-8 se obtendrá el siguiente comportamiento:

- DIP 1-8 a ON: la dirección obtenida se almacenará y no se volverá a solicitar una IP al servidor BootP. La IP puede ser borrada restaurando los parámetros por defecto.
- DIP 1-8 a OFF: la dirección será válida hasta que se apague el módulo, por lo que en el siguiente arranque se volverá a solicitar una nueva IP.

1.5.5.1.5.1 Ejemplo de asignación automática mediante el programa TFtd32

Para poder realizar dicha asignación, disponemos del servidor *TFtpd32*, que se instala en el directorio c:\Galileo\ConfigModBus. Es necesario que esté ejecutándose en el PC desde el que se configuran los módulos. Este servidor no requiere instalación y se ejecuta simplemente haciendo doble clic sobre su icono. El procedimiento para configurar los distintos módulos sería el siguiente:

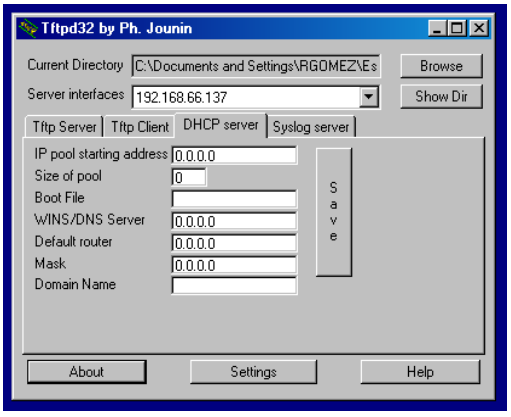
Primero se le asigna una dirección IP que nos interese al PC.

A partir de esa dirección, se le empezarán a asignar a los módulos Beckhoff direcciones consecutivas.

La forma más segura de dar las direcciones IP a los módulos Beckhoff es hacerlo módulo por módulo, empezando por el módulo del Armario C0.

Los pasos son los siguientes:

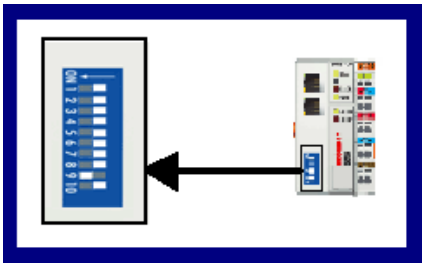
1. Una vez tengamos corriendo el servidor, tendremos que ir a la pestaña DHCP Server y realizar la siguiente configuración:



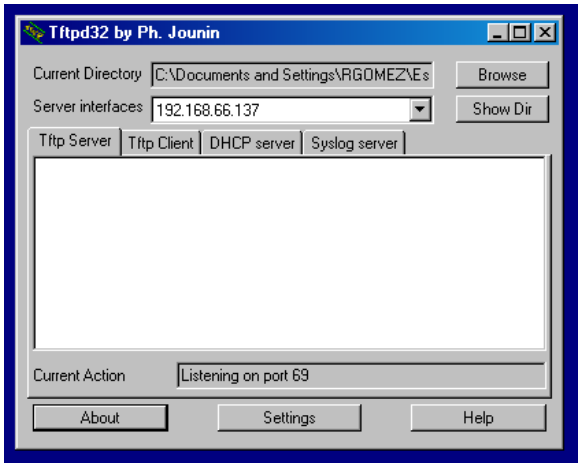
- *IP pool starting address*: dirección IP a partir de la cual se asignarán direcciones. Aquí pondremos la siguiente IP que la que le asignamos a nuestro ordenador.
- *Size of pool*: máximo número de IP a dar. En nuestro caso, como sólo vamos a configurar un módulo, le ponemos 1.
- *Mask*: máscara de subred: la misma máscara que se pone al configurar la red. En nuestro caso, 255.255.0.0

- Una vez configurado, es necesario salvar estos datos (botón "save") y reiniciar el servidor.

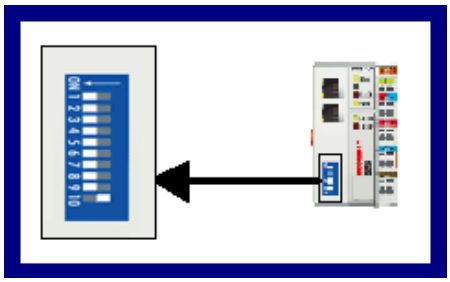
2. Con la tensión de 24 quitada en todos los módulos E/S, en el módulo que vamos a configurar primero (el del Armario C0), ponemos todos los switches del a OFF, excepto el 9 (ver figura).



En este momento, con el servidor corriendo, si se mete tensión al módulo, éste demandará una dirección IP, y veremos la dirección propuesta por el servidor en la pestaña "Tftp Server":



3. Una vez que se ha producido el mensaje con la IP propuesta, hay que volver a quitarle tensión de nuevo al módulo, y colocar los switches todos a ON, excepto el 10 (ver figura):



Metiéndole tensión en este momento, veremos de nuevo los mensajes de la IP propuesta en la pestaña "Tftp Server". El módulo Beckhoff ya se queda con esa IP.

Para los demás módulos hay que seguir los mismos pasos, cambiando sólo la dirección que se pone en "IP pool starting address", para que a cada base le vaya asignando la IP siguiente a la que ya hemos utilizado.

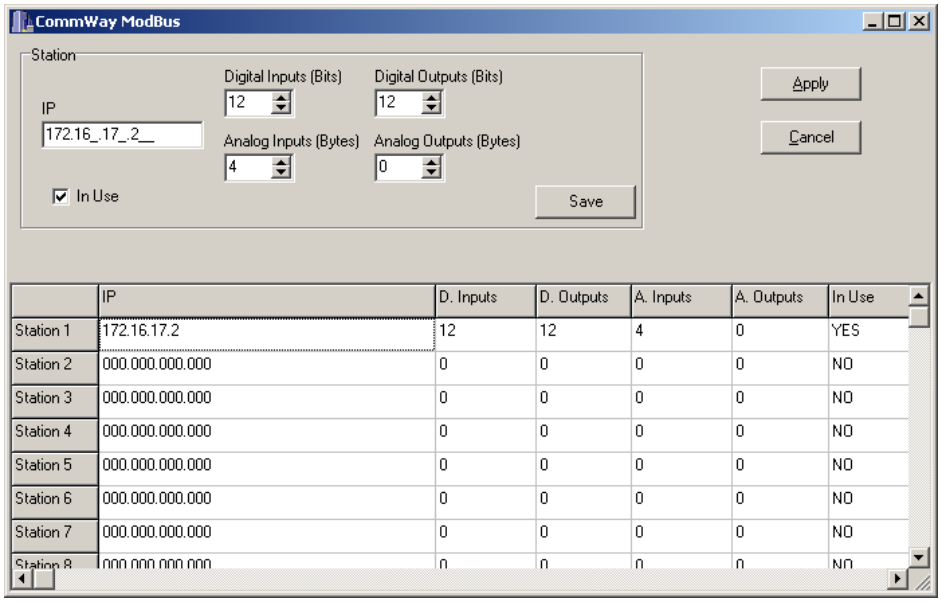
4. Una vez configurados los módulos E/S, es conveniente probar que todos los módulos son accesibles. Para ello, les hacemos un ping y esperamos su respuesta. En la imagen se muestra un ejemplo de ping para un módulo al que se le asignó la IP 192.168.66162:

1.5.5.1.6 Activación de los módulos

Una vez se tengan configurados los distintos módulos es necesario darles un orden para su uso en Designer. Para ello se utiliza la aplicación *commwaymb.exe*, ubicada en C:\Galileo\ConfigModBus.

En esta aplicación definiremos, para cada módulo, su IP, su número de *bits* de entradas y salidas digitales, el numero de *bytes* de entradas y salidas analógicas, y un índice que indicará su número de estación, así como si está o no activa (in use).

Para ello, hay que hacer click en el nº de estación que se quiere definir, y cubrir los datos en el combobox "Station". Para que los cambios tengan efecto, primero hay que salvar (Save) y después aplicar (Apply).



Es importante tener en cuenta que independientemente de como están colocados los módulos en la cabecera, la imagen de proceso mostrará las entradas/salidas analógicas en las direcciones mas bajas del bus de ese elemento.

1.6 Comunicaciones TCP/IP

1.6.1 Introducción

Internet no es un nuevo tipo de red física, sino un conjunto de tecnologías que permiten interconectar redes muy distintas entre sí. Internet no es dependiente de la máquina ni del sistema operativo utilizado. De esta manera, podemos transmitir información entre un servidor Unix y un ordenador que utilice Windows. O entre plataformas completamente distintas como Macintosh, Alpha o Intel. Es más, entre una máquina y otra generalmente existirán redes distintas: redes Ethernet, redes Token Ring e incluso enlaces vía satélite.

Como vemos, está claro que no podemos utilizar ningún protocolo que dependa de una arquitectura en particular. Lo que estamos buscando es un método de interconexión general que sea válido para cualquier plataforma, sistema operativo y tipo de red. La familia de protocolos que se eligieron para permitir que Internet sea una Red de redes es TCP/IP. Nótese aquí que hablamos de familia de protocolos ya que son muchos los protocolos que la integran, aunque en ocasiones para simplificar hablemos sencillamente del protocolo TCP/IP.

El protocolo TCP/IP tiene que estar a un nivel superior del tipo de red empleado y funcionar de forma transparente en cualquier tipo de red. Y a un nivel inferior de los programas de aplicación (páginas WEB, correo electrónico…) particulares de cada sistema operativo. Todo esto nos sugiere el siguiente modelo de referencia:

Capa de aplicación (HTTP, SMTP, FTP, TELNET...)
Capa de transporte (UDP, TCP)
Capa de red (IP)
Capa de acceso a la red (Ethernet, Token Ring...)
Capa física (cable coaxial, par trenzado...)

El nivel más bajo es la capa física. Aquí nos referimos al medio físico por el cual se transmite la información. Generalmente será un cable aunque no se descarta cualquier otro medio de transmisión como ondas o enlaces vía satélite.
La capa de acceso a la red determina la manera en que las estaciones (ordenadores) envían y reciben la información a través del soporte físico proporcionado por la capa anterior. Es decir, una vez que tenemos un cable, ¿cómo se transmite la información por ese cable? ¿Cuándo puede una estación transmitir? ¿Tiene que esperar algún turno o transmite sin más? ¿Cómo sabe una estación que un mensaje es para ella? Pues bien, son todas estas cuestiones las que resuelve esta capa.

Las dos capas anteriores quedan a un nivel inferior del protocolo TCP/IP, es decir, no forman parte de este protocolo. La capa de red define la forma en que un mensaje se transmite a través de distintos tipos de redes hasta llegar a su destino. El principal protocolo de esta capa es el IP aunque también se encuentran a este nivel los protocolos ARP, ICMP e IGMP. Esta capa proporciona el direccionamiento IP y determina la ruta óptima a través de los encaminadores (routers) que debe seguir un paquete desde el origen al destino.

La capa de transporte (protocolos TCP y UDP) ya no se preocupa de la ruta que siguen los mensajes hasta llegar a su destino. Sencillamente, considera que la comunicación extremo a extremo está establecida y la utiliza. Además añade la noción de puertos, como veremos más adelante.

Una vez que tenemos establecida la comunicación desde el origen al destino nos queda lo más importante, ¿qué podemos transmitir? La capa de aplicación nos proporciona los distintos servicios de Internet: correo electrónico, páginas Web, FTP, TELNET…

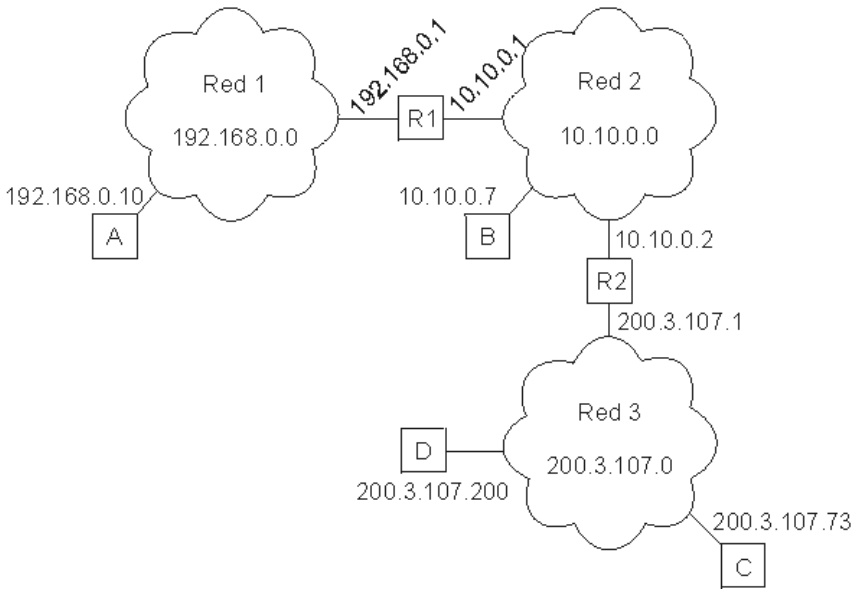
1.6.2 Capa de red

La familia de protocolos TCP/IP fue diseñada para permitir la interconexión entre distintas redes. El mejor ejemplo de interconexión de redes es Internet: se trata de un conjunto de redes unidas mediante encaminadores o routers.

A lo largo de esta introducción explicaremos como construir redes privadas que funcionen siguiendo el mismo esquema de Internet. En una red TCP/IP es posible tener, por ejemplo, servidores web y servidores de correo para uso interno. Obsérvese que todos los servicios de Internet se pueden configurar en pequeñas redes internas TCP/IP.

A continuación veremos un ejemplo de interconexión de 3 redes. Cada host (ordenador) tiene una dirección física que viene determinada por su adaptador de red. Estas direcciones se corresponden con la capa de acceso al medio y se utilizan para comunicar dos ordenadores que pertenecen a la misma red. Para identificar globalmente un ordenador dentro de un conjunto de redes TCP/IP se utilizan las direcciones IP (capa de red). Observando una dirección IP sabremos si pertenece a nuestra propia red o a una distinta (todas las direcciones IP de la misma red comienzan con los mismos números, según veremos más adelante).

Host	Dirección física	Dirección IP	Red
A	00-60-52-0B-B7-7D	192.168.0.10	Red 1
R1	00-E0-4C-AB-9A-FF	192.168.0.1	
	A3-BB-05-17-29-D0	10.10.0.1	Red 2
B	00-E0-4C-33-79-AF	10.10.0.7	
R2	B2-42-52-12-37-BE	10.10.0.2	
	00-E0-89-AB-12-92	200.3.107.1	Red 3
C	A3-BB-08-10-DA-DB	200.3.107.73	
D	B2-AB-31-07-12-93	200.3.107.200	



El concepto de red está relacionado con las direcciones IP que se configuren en cada ordenador, no con el cableado. Es decir, si tenemos varias redes dentro del mismo cableado solamente los ordenadores que permanezcan a una misma red podrán comunicarse entre sí. Para que los ordenadores de una red puedan comunicarse con los de otra red es necesario que existan routers que interconecten las redes. Un router o encaminador no es más que un ordenador con varias direcciones IP, una para cada red, que permita el tráfico de paquetes entre sus redes.

La capa de red se encarga de fragmentar cada mensaje en paquetes de datos llamados datagramas IP y de enviarlos de forma independiente a través de la red

de redes. Cada datagrama IP incluye un campo con la dirección IP de destino. Esta información se utiliza para enrutar los datagramas a través de las redes necesarias que los hagan llegar hasta su destino.

NOTA: *Cada vez que visitamos una página web o recibimos un correo electrónico es habitual atravesar un número de redes comprendido entre 10 y 20, dependiendo de la distancia de los hosts. El tiempo que tarda un datagrama en atravesar 20 redes (20 routers) suele ser inferior a 600 milisegundos.*

En el ejemplo anterior, supongamos que el ordenador 200.3.107.200 (D) envía un mensaje al ordenador con 200.3.107.73 (C). Como ambas direcciones comienzan con los mismos números, D sabrá que ese ordenador se encuentra dentro de su propia red y el mensaje se entregará de forma directa. Sin embargo, si el ordenador 200.3.107.200 (D) tuviese que comunicarse con 10.10.0.7 (B), D advertiría que el ordenador destino no pertenece a su propia red y enviaría el mensaje al router R2 (es el ordenador que le da salida a otras redes). El router entregaría el mensaje de forma directa porque B se encuentra dentro de una de sus redes (la Red 2).

1.6.3 Direcciones IP

La dirección IP es el identificador de cada host dentro de su red de redes. Cada host conectado a una red tiene una dirección IP asignada, la cual debe ser distinta a todas las demás direcciones que estén vigentes en ese momento en el conjunto de redes visibles por el host. En el caso de Internet, no puede haber dos ordenadores con 2 direcciones IP (públicas) iguales. Pero sí podríamos tener dos ordenadores con la misma dirección IP siempre y cuando pertenezcan a redes independientes entre sí (sin ningún camino posible que las comunique).

Las direcciones IP se clasifican en:

- *Direcciones IP públicas*. Son visibles en todo Internet. Un ordenador con una IP pública es accesible (visible) desde cualquier otro ordenador conectado a Internet. Para conectarse a Internet es necesario tener una dirección IP pública.
- *Direcciones IP privadas (reservadas).* Son visibles únicamente por otros hosts de su propia red o de otras redes privadas interconectadas por routers. Se utilizan en las empresas para los puestos de trabajo. Los ordenadores con direcciones IP privadas pueden salir a Internet por medio de un router (o proxy) que tenga una IP pública. Sin embargo, desde Internet no se puede acceder a ordenadores con direcciones IP privadas.

A su vez, las direcciones IP pueden ser:

- *Direcciones IP estáticas (fijas).* Un host que se conecte a la red con dirección IP estática siempre lo hará con una misma IP. Las direcciones IP públicas estáticas son las que utilizan los servidores de Internet con objeto de que estén siempre localizables por los usuarios de Internet. Estas direcciones hay que contratarlas.

- *Direcciones IP dinámicas*. Un host que se conecte a la red mediante dirección IP dinámica, cada vez lo hará con una dirección IP distinta. Las direcciones IP públicas dinámicas son las que se utilizan en las conexiones a Internet mediante un módem. Los proveedores de Internet utilizan direcciones IP dinámicas debido a que tienen más clientes que direcciones IP (es muy improbable que todos se conecten a la vez).

Las direcciones IP están formadas por 4 `bytes` (32 `bits`). Se suelen representar de la forma a.b.c.d donde cada una de estas letras es un número comprendido entre el 0 y el 255. Por ejemplo la dirección IP del servidor de IBM (www.ibm.com) es 129.42.18.99.

Las direcciones IP también se pueden representar en hexadecimal, desde la:
00.00.00.00
hasta la:
FF.FF.FF.FF
o en binario, desde la:
00000000.00000000.00000000.00000000
hasta la:
11111111.11111111.11111111.11111111.

Las tres direcciones siguientes representan a la misma máquina.

(decimal)	128.10.2.30
(hexadecimal)	80.0A.02.1E
(binario)	10000000.00001010.00000010.00011110

¿Cuántas direcciones IP existen? Si calculamos 2 elevado a 32 obtenemos más de 4000 millones de direcciones distintas. Sin embargo, no todas las direcciones son válidas para asignarlas a hosts. Las direcciones IP no se encuentran aisladas en Internet, sino que pertenecen siempre a alguna red. Todas las máquinas conectadas a una misma red se caracterizan en que los primeros bits de sus direcciones son iguales. De esta forma, las direcciones se dividen conceptualmente en dos partes: el identificador de red y el identificador de host.

Dependiendo del número de hosts que se necesiten para cada red, las direcciones de Internet se han dividido en las clases primarias A, B y C. La clase D está formada por direcciones que identifican no a un host, sino a un grupo de ellos. Las direcciones de clase E no se pueden utilizar (están reservadas).

<table>
<tr><td></td><td colspan="5">0 1 2 3 4</td><td colspan="3">8</td><td colspan="3">16</td><td colspan="3">24</td><td>31</td></tr>
<tr><td>Clase A</td><td>0</td><td colspan="4">red</td><td colspan="8">host</td></tr>
<tr><td colspan="14"></td></tr>
<tr><td>Clase B</td><td>1</td><td>0</td><td colspan="6">red</td><td colspan="6">host</td></tr>
<tr><td colspan="14"></td></tr>
<tr><td>Clase C</td><td>1</td><td>1</td><td>0</td><td colspan="8">red</td><td colspan="3">host</td></tr>
<tr><td colspan="14"></td></tr>
<tr><td>Clase D</td><td>1</td><td>1</td><td>1</td><td>0</td><td colspan="10">grupo de multicast (multidifusión)</td></tr>
<tr><td colspan="14"></td></tr>
<tr><td>Clase E</td><td>1</td><td>1</td><td>1</td><td>1</td><td colspan="10">(direcciones reservadas: no se pueden utilizar)</td></tr>
</table>

Clase	Formato (r=red, h=host)	Número de redes	Número de hosts por red	Rango de direcciones de redes	Máscara de subred
A	r.h.h.h	128	16.777.214	0.0.0.0 - 127.0.0.0	255.0.0.0
B	r.r.h.h	16.384	65.534	128.0.0.0 - 191.255.0.0	255.255.0.0
C	r.r.r.h	2.097.152	254	192.0.0.0 - 223.255.255.0	255.255.255.0
D	grupo	-	-	224.0.0.0 - 239.255.255.255	-
E	no válidas	-	-	240.0.0.0 - 255.255.255.255	-

NOTA: Las *direcciones usadas en Internet están definidas en la RFC 1166 (en inglés).*

- *Difusión (broadcast) y multidifusión (multicast).*-- El término difusión (broadcast) se refiere a todos los hosts de una red; multidifusión (multicast) se refiere a varios hosts (aquellos que se hayan suscrito dentro de un mismo grupo). Siguiendo esta misma terminología, en ocasiones se utiliza el término unidifusión para referirse a un único host.

1.6.4 Direcciones IP especiales y reservadas

No todas las direcciones comprendidas entre la 0.0.0.0 y la 255.255.255.255 son válidas para un host: algunas de ellas tienen significados especiales. Las principales direcciones especiales se resumen en la siguiente tabla. Su interpretación depende del host desde el que se utilicen.

Bits de red	Bits de host	Significado	Ejemplo
todos 0		Mi propio host	0.0.0.0
todos 0	host	Host indicado dentro de mi red	0.0.0.10
red	todos 0	Red indicada	192.168.1.0
todos 1		Difusión a mi red	255.255.255.255
red	todos 1	Difusión a la red indicada	192.168.1.255
127	cualquier valor válido de host	Loopback (mi propio host)	127.0.0.1

Difusión o broadcasting es el envío de un mensaje a todos los ordenadores que se encuentran en una red. La dirección de loopback (normalmente 127.0.0.1) se utiliza para comprobar que los protocolos TCP/IP están correctamente instalados en nuestro propio ordenador.

Las direcciones de redes siguientes se encuentran reservadas para su uso en redes privadas (intranets). Una dirección IP que pertenezca a una de estas redes se dice que es una dirección IP privada.

Clase	Rango de direcciones reservadas de redes
A	10.0.0.0
B	172.16.0.0 - 172.31.0.0
C	192.168.0.0 - 192.168.255.0

Intranet.-- Red privada que utiliza los protocolos TCP/IP. Puede tener salida a Internet o no. En el caso de tener salida a Internet, el direccionamiento IP permite que los hosts con direcciones IP privadas puedan salir a Internet pero impide el acceso a los hosts internos desde Internet. Dentro de una intranet se pueden configurar todos los servicios típicos de Internet (web, correo, mensajería instantánea, etc.) mediante la instalación de los correspondientes servidores. La idea es que las intranets son como "internets" en miniatura o lo que es lo mismo, Internet es una intranet pública gigantesca.

Extranet.-- Unión de dos o más intranets. Esta unión puede realizarse mediante líneas dedicadas (RDSI, X.25, frame relay, punto a punto, etc.) o a través de Internet.

Internet.-- La mayor red pública de redes TCP/IP.

Por ejemplo, si estamos construyendo una red privada con un número de ordenadores no superior a 254 podemos utilizar una red reservada de clase C. Al primer ordenador le podemos asignar la dirección 192.168.23.1, al segundo 192.168.23.2 y así sucesivamente hasta la 192.168.23.254. Como estamos utilizando direcciones reservadas, tenemos la garantía de que no habrá ninguna máquina conectada directamente a Internet con alguna de nuestras direcciones. De esta manera, no se producirán conflictos y desde cualquiera de nuestros ordenadores podremos acceder a la totalidad de los servidores de Internet (si utilizásemos en un ordenador de nuestra red una dirección de un servidor de Internet, nunca podríamos acceder a ese servidor).

1.6.5 Máscara de subred

Una máscara de subred es aquella dirección que enmascarando nuestra dirección IP, nos indica si otra dirección IP pertenece a nuestra subred o no.

La siguiente tabla muestra las máscaras de subred correspondientes a cada clase:

Clase	Máscara de subred
A	255.0.0.0
B	255.255.0.0
C	255.255.255.0

Si expresamos la máscara de subred de clase A en notación binaria, tenemos:
11111111.00000000.00000000.00000000

Los "1" indican los bits de la dirección correspondientes a la red y los ceros, los correspondientes al host. Según la máscara anterior, el primer byte (8 bits) es la red y los tres siguientes (24 bits), el host.
Por ejemplo, la dirección de clase A 35.120.73.5 pertenece a la red 35.0.0.0.

Supongamos una subred con máscara 255.255.0.0, en la que tenemos un ordenador con dirección 148.120.33.110. Si expresamos esta dirección y la de la máscara de subred en binario, tenemos:

```
148.120.33.110 10010100.01111000.00100001.01101110 (dirección de una
máquina)
255.255.0.0    11111111.11111111.00000000.00000000 (dirección de su
máscara de red)
148.120.0.0    10010100.01111000.00000000.00000000 (dirección de su
subred)
               <------RED------> <------HOST------>
```

Al hacer el producto binario de las dos primeras direcciones (donde hay dos 1 en las mismas posiciones ponemos un 1 y en caso contrario, un 0) obtenemos la tercera.
Si hacemos lo mismo con otro ordenador, por ejemplo el 148.120.33.89, obtenemos la misma dirección de subred. Esto significa que ambas máquinas se encuentran en la misma subred (la subred 148.120.0.0).

```
148.120.33.89 10010100.01111000.00100001.01011001 (dirección de una
máquina)
255.255.0.0   11111111.11111111.00000000.00000000 (dirección de su
máscara de red)
148.120.0.0   10010100.01111000.00000000.00000000 (dirección de su
subred)
```

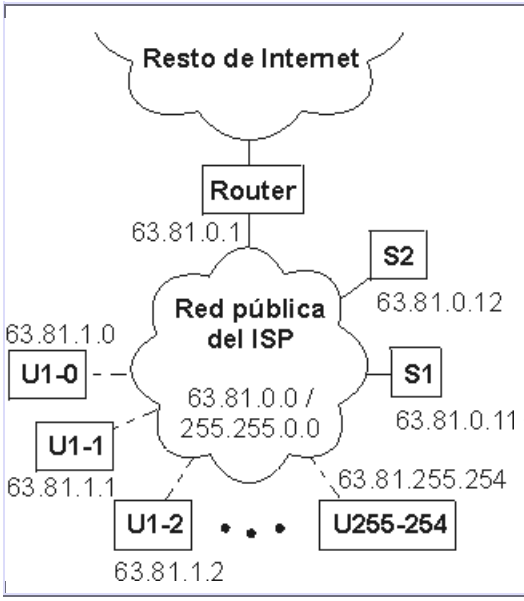
En cambio, si tomamos la 148.115.89.3, observamos que no pertenece a la misma subred que las anteriores.

```
148.115.89.3   10010100.01110011.01011001.00000011 (dirección de una
máquina)
255.255.0.0    11111111.11111111.00000000.00000000 (dirección de su
máscara de red)
148.115.0.0    10010100.01110011.00000000.00000000 (dirección de su
subred)
```

Cálculo de la dirección de difusión.- Ya hemos visto que el producto lógico binario (AND) de una IP y su máscara devuelven su dirección de red. Para calcular su dirección de difusión, hay que hacer la suma lógica en binario (OR) de la IP con el inverso (NOT) de su máscara.

En una red de redes TCP/IP no puede haber hosts aislados: todos pertenecen a alguna red y todos tienen una dirección IP y una máscara de subred (si no se especifica se toma la máscara que corresponda a su clase). Mediante esta máscara un ordenador sabe si otro ordenador se encuentra en su misma subred o en otra distinta. Si pertenece a su misma subred, el mensaje se entregará directamente. En cambio, si los hosts están configurados en redes distintas, el mensaje se enviará a la puerta de salida o router de la red del host origen. Este router pasará el mensaje al siguiente de la cadena y así sucesivamente hasta que se alcance la red del host destino y se complete la entrega del mensaje.

EJEMPLO.- Los proveedores de Internet habitualmente disponen de una o más redes públicas para dar acceso a los usuarios que se conectan por módem. El proveedor va cediendo estas direcciones públicas a sus clientes a medida que se conectan y liberándolas según se van desconectando (direcciones dinámicas).



Supongamos que cierto ISP (proveedor de servicios de Internet) dispone de la red 63.81.0.0 con máscara 255.255.0.0. Para uso interno utiliza las direcciones que comienzan por 63.81.0 y para ofrecer acceso a Internet a sus usuarios, las direcciones comprendidas entre la 63.81.1.0 hasta la 63.81.255.254 (las direcciones 63.81.0.0 y 63.81.255.255 están reservadas).
Si un usuario conectado a la red de este ISP tiene la dirección 63.81.1.1 y quiere transferir un archivo al usuario con IP 63.81.1.2, el primero advertirá que el destinatario se encuentra en su misma subred y el mensaje no saldrá de la red del proveedor (no atravesará el router).

Las máscaras 255.0.0.0 (clase A), 255.255.0.0 (clase B) y 255.255.255.0 (clase C) suelen ser suficientes para la mayoría de las redes privadas. Sin embargo, las redes más pequeñas que podemos formar con estas máscaras son de 254 hosts y para el caso de direcciones públicas, su contratación tiene un coste muy alto. Por esta razón suele ser habitual dividir las redes públicas de clase C en subredes más pequeñas. A continuación se muestran las posibles divisiones de una red de clase C. La división de una red en subredes se conoce como subnetting.

Máscara de subred	*Binario*	*Número de subredes*	*Núm. de hosts por subred*	*Ejemplos de subredes (x=a.b.c por ejemplo, 192.168.1)*
255.255.255.0	00000000	1	254	x.0
255.255.255.128	10000000	2	126	x.0, x.128
255.255.255.192	11000000	4	62	x.0, x.64, x.128, x.192
255.255.255.224	11100000	8	30	x.0, x.32, x.64, x.96, x.128, ...
255.255.255.240	11110000	16	14	x.0, x.16, x.32, x.48, x.64, ...
255.255.255.248	11111000	32	6	x.0, x.8, x.16, x.24, x.32, x.40, ...
255.255.255.252	11111100	64	2	x.0, x.4, x.8, x.12, x.16, x.20, ...
255.255.255.254	11111110	128	0	ninguna posible
255.255.255.255	11111111	256	0	ninguna posible

Obsérvese que en el caso práctico que explicamos un poco más arriba se utilizó la máscara 255.255.255.248 para crear una red pública con 6 direcciones de hosts válidas (la primera y última dirección de todas las redes se excluyen). Las máscaras con `bytes` distintos a 0 o 255 también se pueden utilizar para particionar redes de clase A o de clase B, sin embargo no suele ser lo más habitual. Por ejemplo, la máscara 255.255.192.0 dividiría una red de clase B en 4 subredes de 16382 hosts (2 elevado a 14, menos 2) cada una.

1.6.6 Protocolo IP

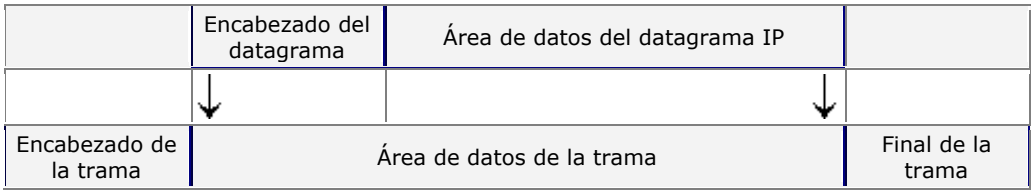
IP es el principal protocolo de la capa de red. Este protocolo define la unidad básica de transferencia de datos entre el origen y el destino, atravesando toda la red de redes. Además, el software IP es el encargado de elegir la ruta más adecuada por la que los datos serán enviados. Se trata de un sistema de entrega de paquetes (llamados datagramas IP) que tiene las siguientes características:

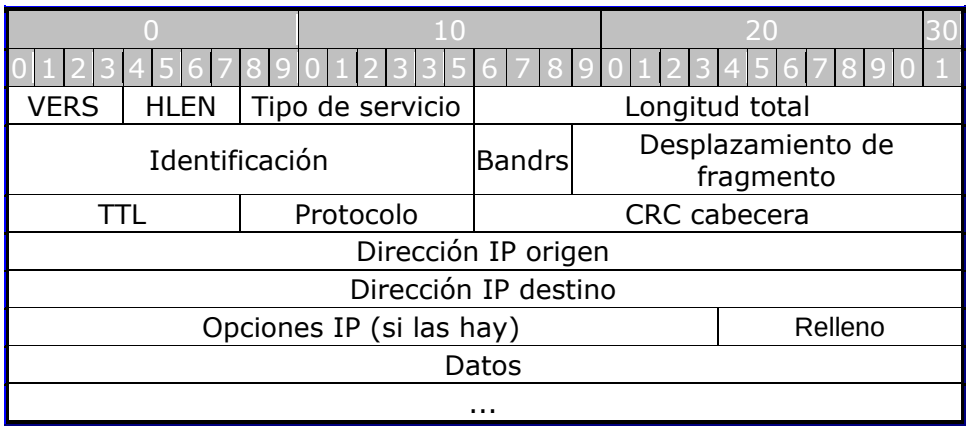
Es no orientado a conexión debido a que cada uno de los paquetes puede seguir rutas distintas entre el origen y el destino. Entonces pueden llegar duplicados o desordenados. Es no fiable porque los paquetes pueden perderse, dañarse o llegar retrasados.

NOTA: El protocolo IP está definido en la RFC 791 (*en inglés*, *en español*).

1.6.6.1 Formato del datagrama IP

El datagrama IP es la unidad básica de transferencia de datos entre el origen y el destino. Viaja en el campo de datos de las tramas físicas (recuérdese la trama Ethernet) de las distintas redes que va atravesando. Cada vez que un datagrama tiene que atravesar un router, el datagrama saldrá de la trama física de la red que abandona y se acomodará en el campo de datos de una trama física de la siguiente red. Este mecanismo permite que un mismo datagrama IP pueda atravesar redes distintas: enlaces punto a punto, redes ATM, redes Ethernet, redes Token Ring, etc. El propio datagrama IP tiene también un campo de datos: será aquí donde viajen los paquetes de las capas superiores.





Campos del datagrama IP:

- VERS (4 bits). Indica la versión del protocolo IP que se utilizó para crear el datagrama. Actualmente se utiliza la versión 4 (IPv4) aunque ya se están preparando las especificaciones de la siguiente versión, la 6 (IPv6).

- HLEN (4 bits). Longitud de la cabecera expresada en múltiplos de 32 bits. El valor mínimo es 5, correspondiente a 160 bits = 20 bytes.

- Tipo de servicio (Type Of Service). Los 8 bits de este campo se dividen a su vez en:

- Prioridad (3 bits). Un valor de 0 indica baja prioridad y un valor de 7, prioridad máxima.

- Los siguientes tres bits indican cómo se prefiere que se transmita el mensaje, es decir, son sugerencias a los encaminadores que se encuentren a su paso los cuales pueden tenerlas en cuenta o no.

    - Bit D (Delay). Solicita retardos cortos (enviar rápido).
    - Bit T (Throughput). Solicita un alto rendimiento (enviar mucho en el menor tiempo posible).
    - Bit R (Reliability). Solicita que se minimice la probabilidad de que el datagrama se pierda o resulte dañado (enviar bien).

- Los siguientes dos bits no tienen uso.

- Longitud total (16 bits). Indica la longitud total del datagrama expresada en `bytes`. Como el campo tiene 16 bits, la máxima longitud posible de un datagrama será de 65535 `bytes`.

- Identificación (16 bits). Número de secuencia que junto a la dirección origen, dirección destino y el protocolo utilizado identifica de manera única un datagrama en toda la red. Si se trata de un datagrama fragmentado, llevará la misma identificación que el resto de fragmentos.

- Banderas o indicadores (3 bits). Sólo 2 bits de los 3 bits disponibles están actualmente utilizados. El `bit` de Más fragmentos (MF) indica que no es el último datagrama. Y el `bit` de No fragmentar (NF) prohíbe la fragmentación del datagrama. Si este bit está activado y en una determinada red se requiere fragmentar el datagrama, éste no se podrá transmitir y se descartará.

- Desplazamiento de fragmentación (13 bits). Indica el lugar en el cual se insertará el fragmento actual dentro del datagrama completo, medido en unidades de 64 bits. Por esta razón los campos de datos de todos los fragmentos menos el último tienen una longitud múltiplo de 64 bits. Si el paquete no está fragmentado, este campo tiene el valor de cero.

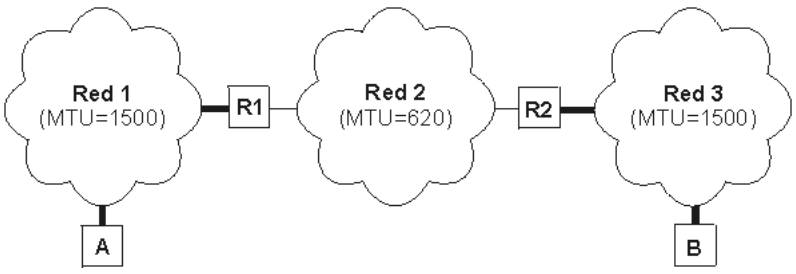
- Tiempo de vida o TTL (8 bits). Número máximo de segundos que puede estar un datagrama en la red de redes. Cada vez que el datagrama atraviesa un router se resta 1 a este número. Cuando llegue a cero, el datagrama se

descarta y se devuelve un mensaje ICMP de tipo "tiempo excedido" para informar al origen de la incidencia.

- Protocolo (8 bits). Indica el protocolo utilizado en el campo de datos: 1 para ICMP, 2 para IGMP, 6 para TCP y 17 para UDP.
- CRC cabecera (16 bits). Contiene la suma de comprobación de errores sólo para la cabecera del datagrama. La verificación de errores de los datos corresponde a las capas superiores.
- Dirección origen (32 bits). Contiene la dirección IP del origen.
- Dirección destino (32 bits). Contiene la dirección IP del destino.
- Opciones IP. Este campo no es obligatorio y especifica las distintas opciones solicitadas por el usuario que envía los datos (generalmente para pruebas de red y depuración).
- Relleno. Si las opciones IP (en caso de existir) no ocupan un múltiplo de 32 bits, se completa con bits adicionales hasta alcanzar el siguiente múltiplo de 32 bits (recuérdese que la longitud de la cabecera tiene que ser múltiplo de 32 bits).

1.6.6.1.1 Fragmentación

Ya hemos visto que las tramas físicas tienen un campo de datos y que es aquí donde se transportan los datagramas IP. Sin embargo, este campo de datos no puede tener una longitud indefinida debido a que está limitado por el diseño de la red. El MTU de una red es la mayor cantidad de datos que puede transportar su trama física. El MTU de las redes Ethernet es 1500 bytes y el de las redes Token-Ring, 8192 bytes. Esto significa que una red Ethernet nunca podrá transportar un datagrama de más de 1500 bytes sin fragmentarlo.
Un encaminador (router) fragmenta un datagrama en varios si el siguiente tramo de la red por el que tiene que viajar el datagrama tiene un MTU inferior a la longitud del datagrama. Veamos con el siguiente ejemplo cómo se produce la fragmentación de un datagrama.



Supongamos que el host A envía un datagrama de 1400 bytes de datos (1420 bytes en total) al host B. El datagrama no tiene ningún problema en atravesar la red 1 ya que 1420 < 1500. Sin embargo, no es capaz de atravesar la red 2 (1420 >= 620). El router R1 fragmenta el datagrama en el menor número de fragmentos posibles que sean capaces de atravesar la red 2. Cada uno de estos fragmentos es

un nuevo datagrama con la misma Identificación pero distinta información en el campo de Desplazamiento de fragmentación y el bit de Más fragmentos (MF). Veamos el resultado de la fragmentación:

- Fragmento 1: Long. total = 620 bytes; Desp = 0; MF=1 (contiene los primeros 600 bytes de los datos del datagrama original)
- Fragmento 2: Long. total = 620 bytes; Desp = 600; MF=1 (contiene los siguientes 600 bytes de los datos del datagrama original)
- Fragmento 3: Long. total = 220 bytes; Desp = 1200; MF=0 (contiene los últimos 200 bytes de los datos del datagrama original)

El router R2 recibirá los 3 datagramas IP (fragmentos) y los enviará a la red 3 sin reensamblarlos. Cuando el host B reciba los fragmentos, recompondrá el datagrama original. Los encaminadores intermedios no reensamblan los fragmentos debido a que esto supondría una carga de trabajo adicional, a parte de memorias temporales. Nótese que el ordenador destino puede recibir los fragmentos cambiados de orden pero esto no supondrá ningún problema para el reensamblado del datagrama original puesto que cada fragmento guarda suficiente información.

Si el datagrama del ejemplo hubiera tenido su bit No fragmentar (NF) a 1, no hubiera conseguido atravesar el router R1 y, por tanto, no tendría forma de llegar hasta el host B. El encaminador R1 descartaría el datagrama.

1.6.7 Protocolo ARP

Dentro de una misma red, las máquinas se comunican enviándose tramas físicas. Las tramas Ethernet contienen campos para las direcciones físicas de origen y destino (6 bytes cada una):

8 bytes	6 bytes	6 bytes	2 bytes	64-1500 bytes	4 bytes
Preámbulo	Dirección física destino	Dirección física origen	Tipo de trama	Datos de la trama	CRC

El problema que se nos plantea es cómo podemos conocer la dirección física de la máquina destino. El único dato que se indica en los datagramas es la dirección IP de destino. ¿Cómo se pueden entregar entonces estos datagramas? Necesitamos obtener la dirección física de un ordenador a partir de su dirección IP. Esta es justamente la misión del protocolo ARP (Address Resolution Protocol, protocolo de resolución de direcciones).

***NOTA**: El protocolo ARP está definido en la RFC 826 (en inglés)*

Host	*Dirección física*	*Dirección IP*	*Red*
A	00-60-52-0B-B7-7D	192.168.0.10	Red 1
R1	00-E0-4C-AB-9A-FF	192.168.0.1	
	A3-BB-05-17-29-D0	10.10.0.1	Red 2
B	00-E0-4C-33-79-AF	10.10.0.7	
R2	B2-42-52-12-37-BE	10.10.0.2	Red 3

	00-E0-89-AB-12-92	200.3.107.1	
C	A3-BB-08-10-DA-DB	200.3.107.73	
D	B2-AB-31-07-12-93	200.3.107.200	

Vamos a retomar el ejemplo introductorio de este Capítulo. El host A envía un datagrama con origen 192.168.0.10 y destino 10.10.0.7 (B). Como el host B se encuentra en una red distinta al host A, el datagrama tiene que atravesar el router 192.168.0.1 (R1). Se necesita conocer la dirección física de R1.

Es entonces cuando entra en funcionamiento el protocolo ARP: A envía un mensaje ARP a todas las máquinas de su red preguntando "¿Cuál es la dirección física de la máquina con dirección IP 192.168.0.1?". La máquina con dirección 192.168.0.1 (R1) advierte que la pregunta está dirigida a ella y responde a A con su dirección física (00-E0-4C-AB-9A-FF). Entonces A envía una trama física con origen 00-60-52-0B-B7-7D y destino 00-E0-4C-AB-9A-FF conteniendo el datagrama (origen 192.168.0.10 y destino 10.10.0.7). Al otro lado del router R2 se repite de nuevo el proceso para conocer la dirección física de B y entregar finalmente el datagrama a B. El mismo datagrama ha viajado en dos tramas físicas distintas, una para la red 1 y otra para la red 2.

Observemos que las preguntas ARP son de difusión (se envían a todas las máquinas). Estas preguntas llevan además la dirección IP y dirección física de la máquina que pregunta. La respuesta se envía directamente a la máquina que formuló la pregunta.

1.6.7.1 Tabla ARP (caché ARP)

Cada ordenador almacena una tabla de direcciones IP y direcciones físicas. Cada vez que formula una pregunta ARP y le responden, inserta una nueva entrada en su tabla. La primera vez que C envíe un mensaje a D tendrá que difundir previamente una pregunta ARP, tal como hemos visto. Sin embargo, las siguientes veces que C envíe mensajes a D ya no será necesario realizar nuevas preguntas puesto que C habrá almacenado en su tabla la dirección física de D. Sin embargo, para evitar incongruencias en la red debido a posibles cambios de direcciones IP o adaptadores de red, se asigna un tiempo de vida de cierto número de segundos a cada entrada de la tabla. Cuando se agote el tiempo de vida de una entrada, ésta será eliminada de la tabla.

Las tablas ARP reducen el tráfico de la red al evitar preguntas ARP innecesarias. Pensemos ahora en distintas maneras para mejorar el rendimiento de la red. Después de una pregunta ARP, el destino conoce las direcciones IP y física del origen. Por lo tanto, podría insertar la correspondiente entrada en su tabla. Pero no sólo eso, sino que todas las estaciones de la red escuchan la pregunta ARP: podrían insertar también las correspondientes entradas en sus tablas. Como es muy probable que otras máquinas se comuniquen en un futuro con la primera, habremos reducido así el tráfico de la red aumentando su rendimiento.

Esto que hemos explicado es para comunicar dos máquinas conectadas a la misma red. Si la otra máquina no estuviese conectada a la misma red, sería necesario atravesar uno o más routers hasta llegar al host destino. La máquina origen, si no la tiene en su tabla, formularía una pregunta ARP solicitando la dirección física del

router y le transferiría a éste el mensaje. Estos pasos se van repitiendo para cada red hasta llegar a la máquina destino.

1.6.8 Trucos TCP/IP útiles para GALILEO

A partir de GALILEO 3.1 se mantiene un alto grado de comunicaciones TCP/IP con todas las computadoras integrantes del proyecto. Esta comunicación se realiza habitualmente a través del nombre de HOST de cada una de las computadoras, por este motivo es necesario tener unos mínimos conocimientos para poder conseguir que las computadoras puedan reconocerse por nombre entre ellas.

En versiones anteriores a Windows XP y 2003 los sistemas Windows utilizaban sobre todo la información de HOST que se transmite a través del protocolo NETBIOS (propietario de Microsoft) y utilizaban WINS (Windows Name Resolution) para hacer las asociaciones HOST -> DIRECCIÓN IP.

Internet siempre ha utilizado DNS (Domain Name Server) para realizar dicha asociación y Microsoft ha seguido esta tendencia en las últimas versiones del Sistemas Operativo (Windows XP y siguientes).

De esta manera debe quedarnos claro que para que GALILEO y DESIGNER se comuniquen entre ellos y que las diferentes computadoras con Sistema de Control Galileo realicen el diálogo entre ellas deben poder resolver todas por nombre de HOST. Para comprobar tal efecto basta realizar un `ping` desde una computadora a las demás computadoras del proyecto. Si no funciona (y sin embargo si nos funciona el PING a la dirección IP de dichas computadoras) el problema está en la resolución del Nombre de Host.

Por ejemplo supongamos un proyecto con las siguientes computadoras:

NOMBRE	*DIRECCIÓN IP*
ENTRADAS	10.0.0.1
SALIDAS	10.0.0.2
TRASLOS	10.0.0.3

Para realizar la prueba correspondiente comenzaremos por colocarnos en la computadora ENTRADAS y abrir una ventana de comandos:

1. Inicio -> Ejecutar -> `CMD`
2. Una vez en la ventana de comandos realizaremos los `ping` correspondientes:
   a. `ping` ENTRADAS
   b. `ping` SALIDAS
   c. `ping` TRASLOS

Si esto no funciona veremos un mensaje indicándonos:
"`Tiempo de espera agotado para esta solicitud`"

Si funciona correctamente nos mostrará las estadísticas de la conexión.

En caso de funcionar repetiremos la comprobación en el resto de computadoras y si también funciona, no hay más tareas a realizar.

En caso de no funcionar, la siguiente operativa es repetir la prueba con las direcciones IP:

1. Inicio → Ejecutar → CMD
2. Una vez en la ventana de comandos realizaremos los PING correspondientes:
   a. `ping` 10.0.0.1
   b. `ping` 10.0.0.2
   c. `ping` 10.0.0.3

En caso de no funcionar el problema es físico de red o de configuración de la tarjeta de red y deberemos realizar las comprobaciones pertinentes para solucionar el problema de cableado / configuración.

En caso de funcionar con la dirección IP el problema está claramente centrado en la resolución de nombres y habrá que arreglarlo antes de intentar que GALILEO o DESIGNER funcionen. Para ello lo primero es ponerse en contacto con personal de informática para comprobar la configuración de DNS, dominios, etc. Si esto no es posible, la solución manual está en realizar una asignación manual HOST → Dirección IP. Para efectuar esta asignación con el explorador de windows abriremos la carpeta:

- Windows (el nombre del directorio en el que windows está instalado)
  - System32 → Drivers → Etc.

En esta carpeta editaremos el archivo HOST que tiene el formato mostrado a continuación:

```
# Copyright (c) 1993-1999 Microsoft Corp.
#
# Éste es un ejemplo de archivo HOSTS usado por Microsoft TCP/IP para Windows.
#
# Este archivo contiene las asignaciones de las direcciones IP a los nombres de
# host. Cada entrada debe permanecer en una línea individual. La dirección IP
# debe ponerse en la primera columna, seguida del nombre de host correspondiente.
# La dirección IP y el nombre de host deben separarse con al menos un espacio.
#
#
# También pueden insertarse comentarios (como éste) en líneas individuales
# o a continuación del nombre de equipo indicándolos con el símbolo "#"
#
# Por ejemplo:
#
#      102.54.94.97     rhino.acme.com          # servidor origen
#       38.25.63.10     x.acme.com              # host cliente x
     127.0.0.1       localhost

     10.0.0.1        entradas

     10.0.0.2        salidas

     10.0.0.3        traslos
```

De esta manera añadiendo manualmente la asignación de dirección IP con nombre de Host solucionaremos los problemas de resolución.

NOTA: *Recordar que si luego se intenta resolver por DNS y seguimos teniendo las entradas alteradas en el archivo de HOST, no será posible tener la respuesta correcta. Una vez alterado el archivo volveremos a realizar los PING a las direcciones de HOST y debe de funcionar correctamente.*

1.7 Comunicaciones inalámbricas

Como se comentó antes, el Sistema de Control Galileo solo funciona con instalaciones realizadas con periferia distribuida. Esto implica que los dispositivos están conectados a un bus en lugar de estarlo directamente al sistema de control (PLC, PC, etc.). Existen distintos protocolos para acceder a este tipo de dispositivos, por ejemplo, Profibus, Interbus, Modbus, etc. El los últimos años han aparecido protocolos nuevos que aprovechan como protocolo de transmisión el estándar TCP/IP, el mas extendido actualmente en el mundo de las comunicaciones. Protocolos como ModBus-TCP utilizan como protocolo de transporte TCP, de manera que se puede implementar sobre redes convencionales. Actualmente, las redes domesticas tienden a utilizar sistemas de comunicaciones inalámbricos. Los protocolos de transporte siguen siendo los mismos, pero el medio de transmisión deja de ser un cable para convertirse en el aire. Existen distintas tecnologías inalámbricas, aunque las más utilizadas actualmente son las tecnologías Wi-fi y la tecnología BlueTooth.

Estos avances están llegando al mundo de la automatización industrial, y algunos fabricantes desarrollan productos que permiten diseñar instalaciones en las que los dispositivos se comunican con el sistema de control mediante enlaces inalámbricos. En principio, estos enlaces son transparentes para el usuario (son protocolos de nivel físico y de enlace), y no debería notar diferencia alguna a un enlace tradicional.

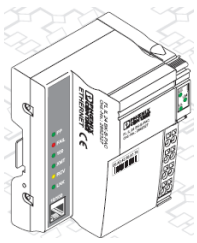
El Sistema de Control Galileo permite trabajar con ciertos sistemas inalámbricos basados en tecnología BlueTooth, por lo que en este apartado se darán las nociones básicas de este estándar.

1.7.1 BlueTooth en el Sistema de Control Galileo

1.7.1.1 Hardware necesario

- Puntos de acceso bluetooth (FL Bluetooth AP)

- Modulo de E/S (FL IL 24 BK-B-PAC)



1.7.1.2 Configuración de las antenas "FL BlueTooth AP"

Login: admin
Password: admin

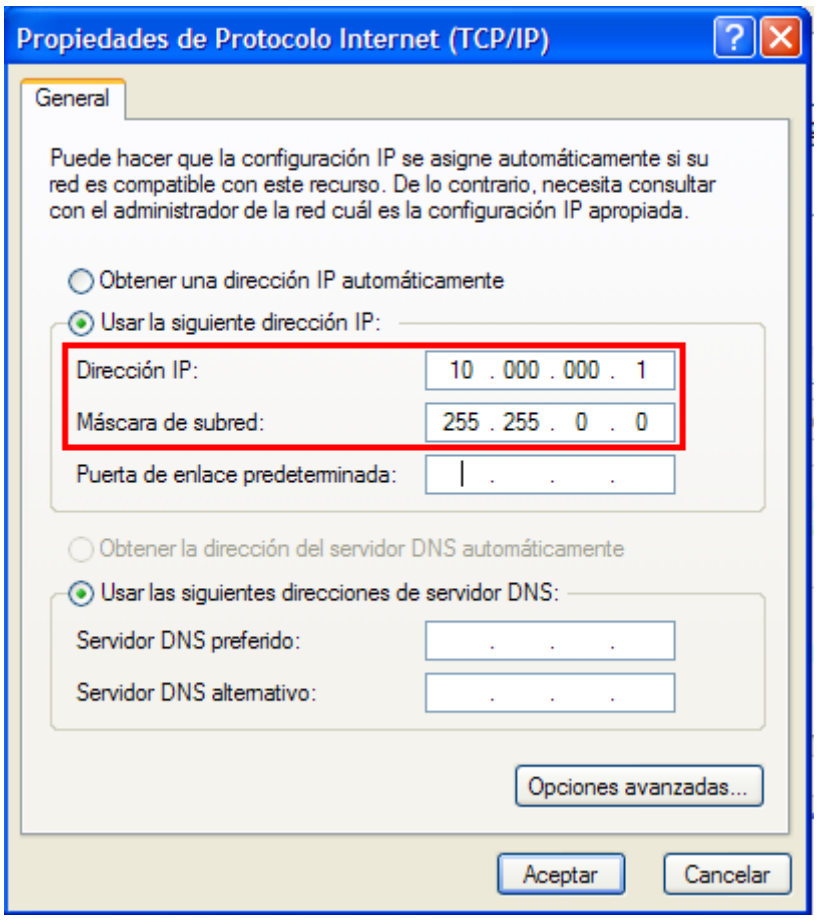
1.7.1.2.1 Configuración del punto de acceso

Lo primero que hay que hacer es configurar los módulos bluetooth para que funcionen en red. Uno de ellos será el que se conecte a la red física (red u ordenador), el punto de acceso, mientras que el resto se conectarán mediante enlaces bluetooth a éste, los clientes.

El primero que hay que conectar es el punto de acceso. Por defecto, la configuración inicial del dispositivo es la siguiente:

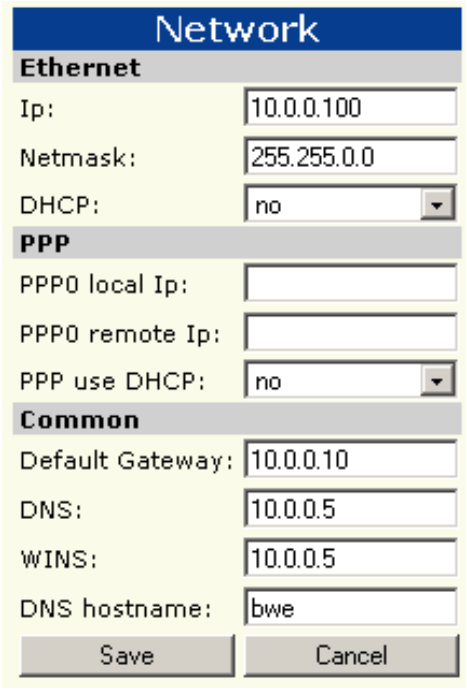
Dirección IP: 10.0.0.100
Mascara: 255.255.0.0

Será necesario para su configuración conectarse a él mediante un cable de red (sin cruzar) y utilizar la aplicación web que incorpora (simplemente escribiendo 10.0.0.100 en el navegador Microsoft Explorer). Es necesario que el ordenador desde el que realice la configuración esté en la misma red que el dispositivo, por lo que, para el caso de la configuración por defecto, se podría utilizar estos valores en el ordenador (IP fija):



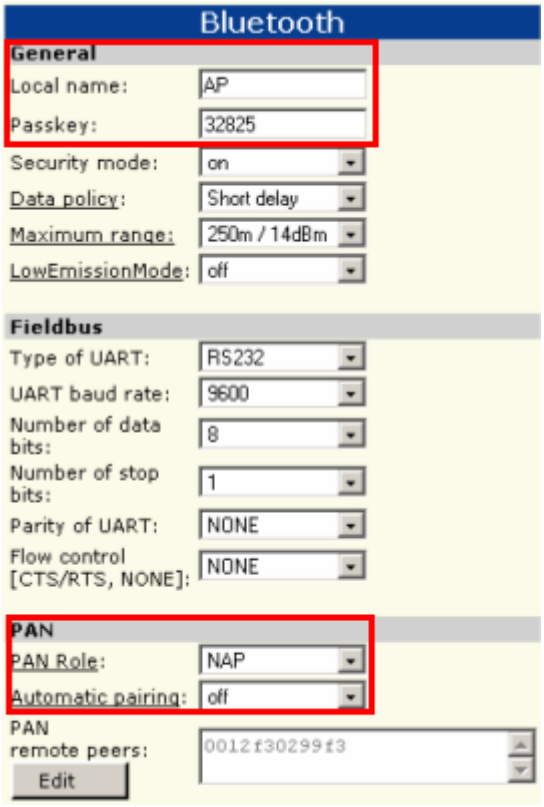
Una vez hecho esto, ya se puede acceder al punto de acceso bluetooth desde un navegador Microsoft Explorer. Para ello simplemente escribimos la dirección del dispositivo en la barra de direcciones del navegador, por ejemplo, para la configuración por defecto: http://10.0.0.100.

Una vez en la aplicación de configuración del dispositivo, pulsamos en la opción ***Network***, que nos permite configurar la red. Para la situación por defecto, se puede utilizar la siguiente configuración:



Es importante tener en cuenta que si se activa la opción DHCP se ignorarán los parámetros IP y Netmask.

El siguiente paso es configurar el enlace bluetooth. Para ello se pulsa sobre la opción ***Bluetooth*** del menú. En la siguiente imagen, se marcan los campos importantes que son necesarios configurar:



- Local name: nombre con el que será conocido el dispositivo en la red bluetooth.
- Passkey: clave necesaria para enlazar los dispositivos bluetooth. Todos los elementos de la red bluetooth tienen que utilizar la misma clave que el punto de acceso.
- PAN role: modo en el que funcionará. Para el caso del punto de acceso tiene que configurarse como NAP y la siguiente opción.
- Automatic pairing: si este parámetro está a *ON*, la antena puede perder la conexión durante unos segundos, buscando otros dispositivos a los que emparejarse, por lo que es necesario poner esta opción a *OFF*, y que solo se conecte a los que se establecerán de manera explicita.

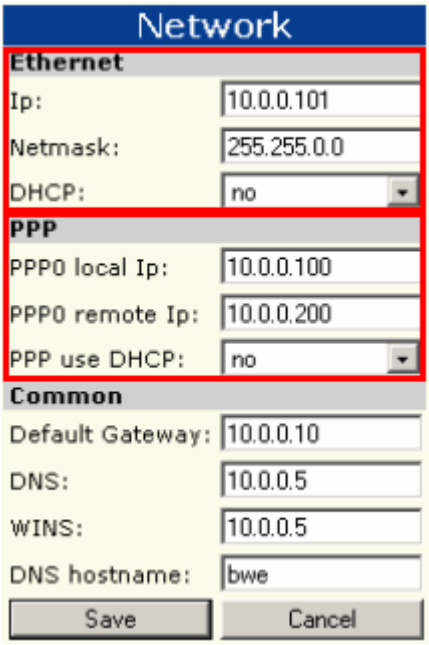
Una vez terminada la configuración, el dispositivo es accesible a través de la red, con dirección 10.0.0.100 y permite crear enlaces bluetooth, por lo que ahora se pasa a la configuración de los clientes.

1.7.1.2.2 Configuración de los clientes

Los pasos para configurar los clientes son similares a los necesarios para configurar el punto de acceso. Inicialmente tienen la misma configuración, es decir:

Dirección IP: 10.0.0.100
Mascara: 255.255.0.0

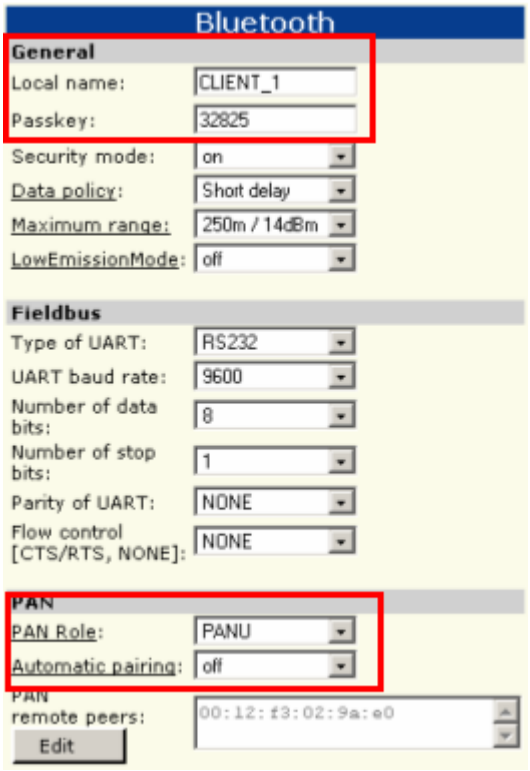
Para conectarnos a ellos seguimos los mismos pasos que para el punto de acceso, es decir, utilizaremos un cable de red sin cruzar y un PC en la misma red. Al igual que con el punto de acceso, se accede a ellos mediante el navegador Internet Explorer y en la opción ***Network*** ponemos los siguientes valores:



En este caso, el de los clientes, tenemos dos partes que configurar. Por un lado esta el grupo etiquetado como Ethernet, que configura las comunicaciones con el punto de acceso antes definido. Para cada cliente que configuremos se utiliza una dirección distinta. Por ejemplo, se podrían utilizar el rango 10.0.0.101-10.0.0.199 para los clientes. Ahora solo resta indicarle la mascara de subred y que no se quiere utilizar DHCP para la asignación de IPs.

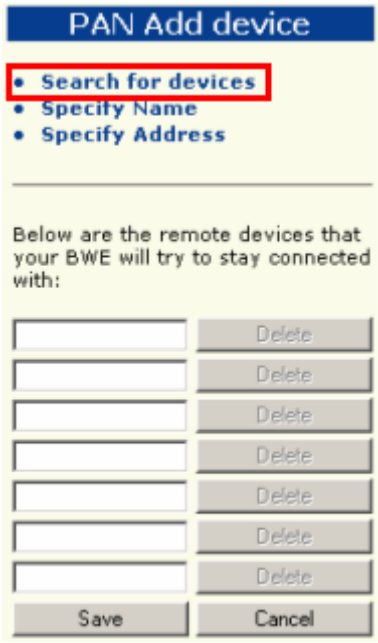
La segunda parte es la comunicación con el módulo a él conectado, en el apartado PPP. Al igual que en el caso anterior no se quiere utilizar el protocolo DHCP. En cuanto a las direcciones IP para el dispositivo local (el módulo bluetooth) y el remoto (el módulo de E/S), como *PPP0 local IP* es necesario poner la dirección del punto de acceso. En *PPP0 remote IP*, a priori, sirve cualquiera, ya que la IP realmente se dará por el protocolo BootP, de todas maneras se puede poner la dirección final que le queremos dar, a modo informativo.

Es necesario ahora, configurar el enlace Bluetooth con el punto de acceso. Para ello, al igual que en el caso del punto de acceso, se pulsa en la opción ***Bluetooth*** del menú. Los parámetros que hay que poner se pueden ver en la siguiente imagen:

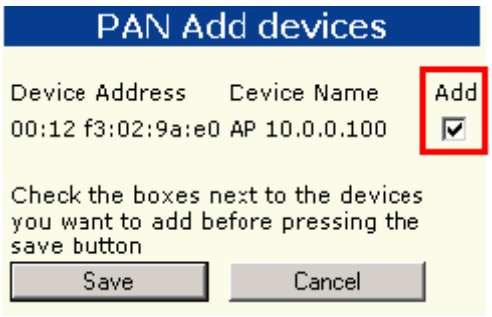


- Local name: nombre con el que será conocido en la red Bluetooth. Cada cliente tiene que tener un nombre distinto.
- Passkey: clave con la que se enlaza al punto de acceso. Tiene que ser la misma que se puso en el punto de acceso.
- PAN Role: modo en el que funciona, en este caso, PANU, y la siguiente opción.
- Automatic pairing: si este parámetro está a on, la antena puede perder la conexión durante unos segundos, buscando otros dispositivos a los que emparejarse, por lo que es necesario poner esta opción a *off*, y que solo se conecte a los que se establecerán de manera explicita.

Ahora es necesario enlazarlo al punto de acceso, de manera que cada vez que se encienda este módulo se establezca el enlace bluetooth de manera automática. Para ello, se pulsa sobre el botón *Edit* marcado en la imagen. Aparecerá un nuevo menú, del que seleccionaremos la opción *Search for devices.*



Aparecerá un nuevo menú con la lista de dispositivos bluetooth al alcance, mostrando su dirección física, su IP y su nombre. Buscamos el que tiene el nombre del punto de acceso que definimos antes, lo marcamos y pulsamos el botón salvar:



Aparecerá entonces el mismo menú de antes, pero con la dirección del punto de acceso en la lista de dispositivos a los que conectarse. Solo resta pulsar el botón *Save* y estará lista la configuración del cliente.

1.7.1.3 Configuración de los modulos de E/S

Password de los módulos:

Solo lectura: *public*
Lectura y escritura: *private*

1.7.1.3.1 Servidor DHCP/Boot

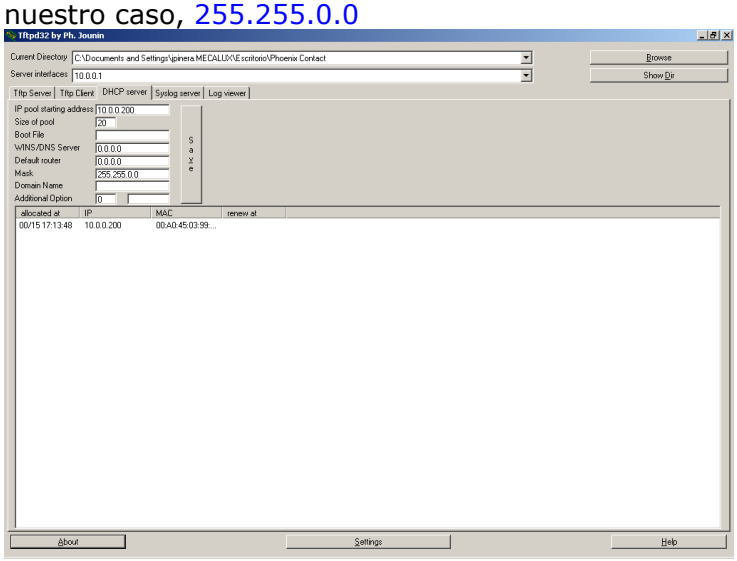
La configuración de los módulos DHCP se realiza en dos pasos. El primero permite asignarle una dirección IP a través de un servidor DHCP/BootP. La segunda parte utiliza el interface web del propio módulo para realizar la configuración definitiva.
Los módulos de E/S obtienen su dirección IP a través del protocolo BootP. Una vez tengan una dirección IP que conozcamos, ya podemos entrar en su configuración y asignarle la que realmente queremos que utilice. Para poder realizar todo esto, es necesario disponer de un servidor de BootP, en nuestro caso TFtpd32, ejecutándose en el ordenador desde el que se configuran los módulos. Este servidor no requiere instalación y simplemente se lanza ejecutando *tftpd32.exe*.

Una vez tengamos corriendo el servidor, tendremos que ir a la pestaña *DHCP Server* y y realizar la siguiente configuración:

- IP pool starting address: dirección IP a partir de la cual se darán direcciones. En el caso de nuestro ejemplo, se asignan direcciones a partir de la 10.0.0.200.

- Size of pool: máximo número de IP a dar. En nuestro caso, configurado a 20, daría direcciones desde 10.0.0.200 a 10.0.0.219.

- Mask: mascara de subred: la misma mascara que se pone al configurar la red. En nuestro caso, 255.255.0.0



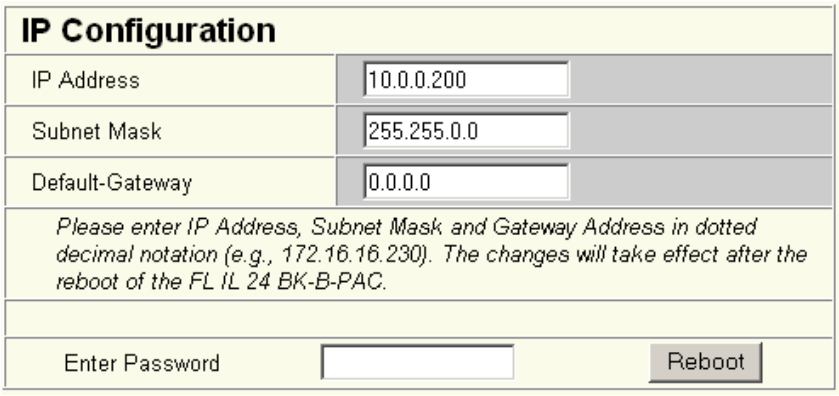
Una vez configurado, es necesario salvar estos datos y reiniciar el servidor. Una vez reiniciado, conectamos los módulos, teniendo en cuenta que es necesario conectar antes el punto de acceso. A medida que se van conectando aparecerán los datos de la conexión en el servidor, en concreto, la MAC del módulo y la IP que se le asigna. Con esta información ya podremos configurar completamente los módulos a través de su interface Web.

1.7.1.3.2 Configuración via WEB

Lo primero que tenemos que hacer es conectar los módulos. En este momento tenemos dos opciones, o bien se conectan en su posición definitiva, es decir mediante un cable de red cruzado a un módulo Bluetooth cliente, o directamente a un PC, también mediante un cable de red cruzado, para su configuración. El proceso de configuración es exactamente igual en las dos formas, así que en esta guía se explicará teniendo como si estuviese conectado en su posición definitiva.

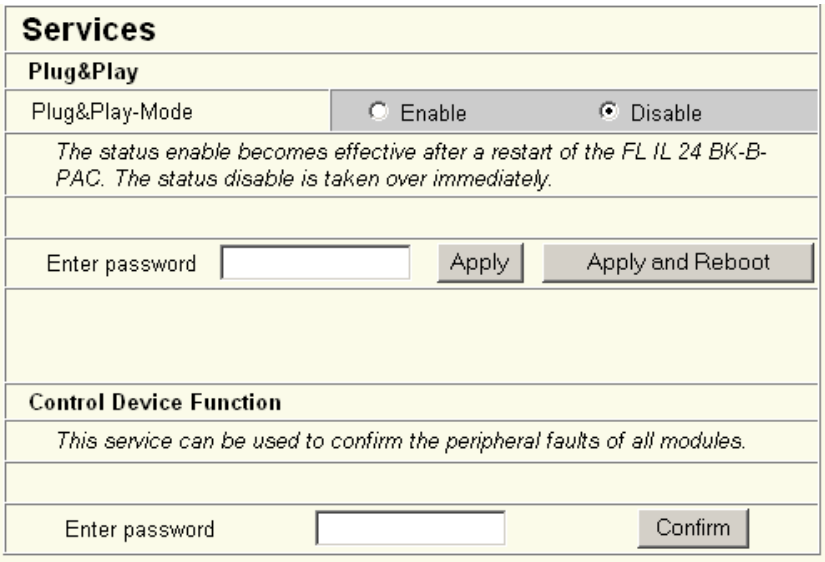
Para configurar los módulos, una vez conocida su IP, basta con escribir esa dirección en el navegador web. Se entrarás así en la aplicación de configuración del módulo. Lo primero que se tiene que hacer es ponerle la dirección IP que realmente queremos (o dejar la que le dio BootP). Para ello, simplemente se pulsa en la *Device Configuration → IP configuration*, y asignarle la IP y la mascara que necesitamos. Hay que tener en cuenta que si el módulo, al arrancar, no encuentra un servidor BootP, utilizará la IP que tiene asignada (que es la última que utilizó).

De manera que sin servidor se comporta como si tuviese IP fija. En el ejemplo, se asigna la IP 10.0.0.200 al módulo y la mascara de nuestra red.

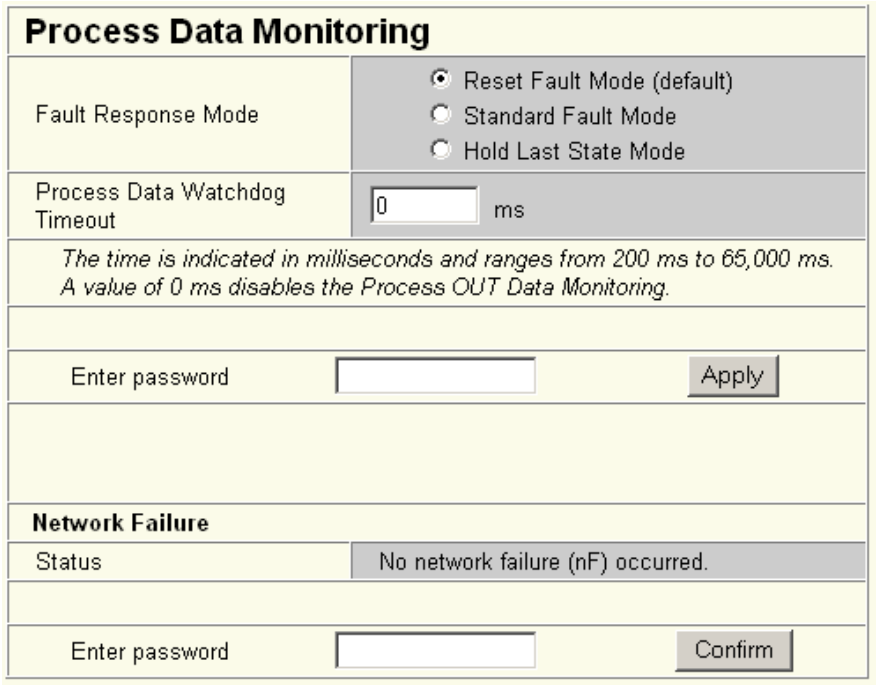


Para aceptar los cambios, es necesario introducir el parsswword y resetear el dispositivo mediante el botón reboot. Después del Reboot, es necesario volver a entrar a la aplicación web, ya que la dirección IP del dispositivo ha cambiado.

Una vez hecho esto, es necesario quitar el modo Plug&Play. Para ello, nos vamos a la opción *Inline Station→Services* del menú de la parte derecha.



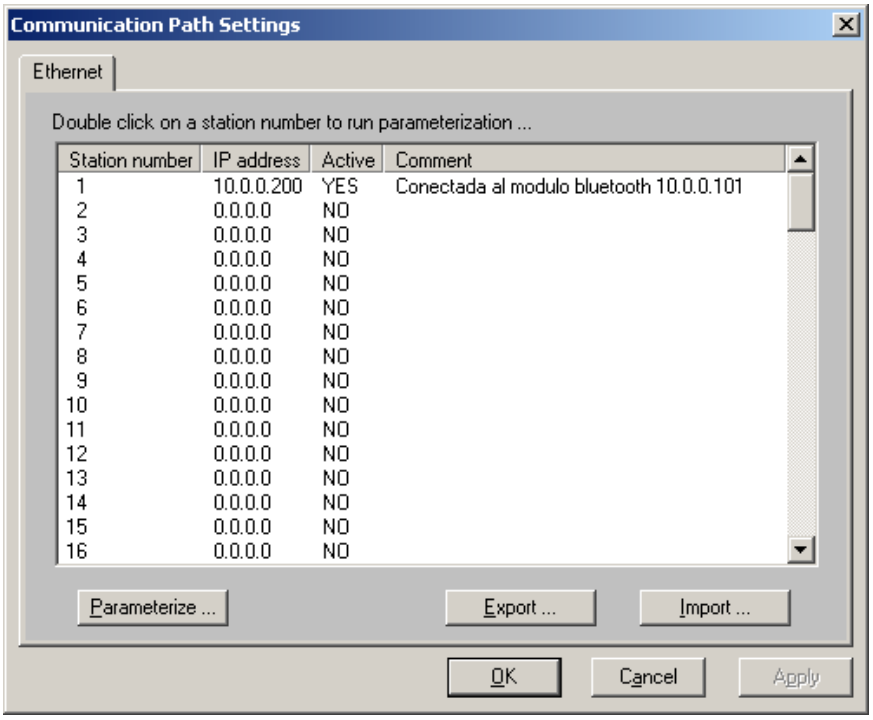
Como en el caso anterior, introducimos el password y aplicamos los cambios antes de abandonar la página. Por último es necesario poner el Watchdog a cero, para evitar que ante ciclos largos falle. Esta opción se controla desde la opción *Inline Station→Process Data Monitoring.*



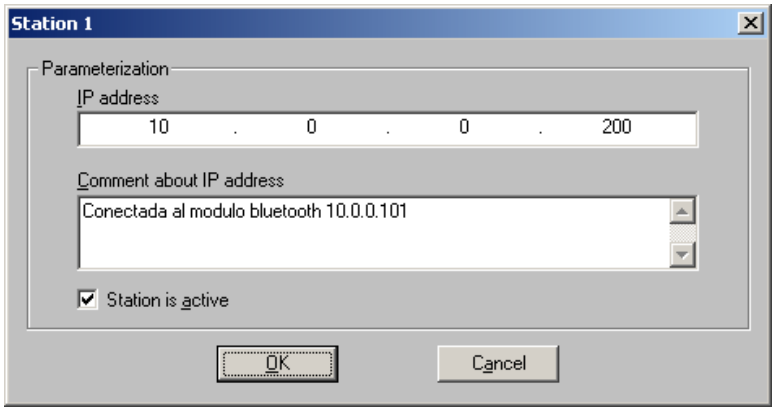
Como siempre, es necesario introducir el password y aplicar los cambios antes de abandonar la página.

1.7.1.3.3 Activación de los módulos

Resta únicamente activar los módulos y asignarles un orden para su uso en Designer. Para ello se utiliza la aplicación *Commway.exe*. En esta aplicación definiremos, para cada estación, su IP y un índice que indicará su número de estación, así como si está o no activa.



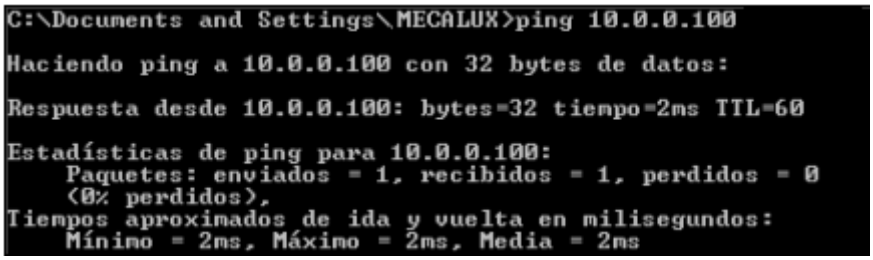
Para ello, basta con hacer doble clic sobre el número de estación que se quiera definir y aparecerá un formulario en el que se introducirán los datos de la IP, el estado (activo o no) y un comentario opcional.



Es importante indicar que para que el sistema funcione correctamente tiene que estar instalado el Driver Ethernet versión 2.0.0.8. Con versiones anteriores de este driver se han detectado problemas de funcionamiento.

1.7.1.3.4 Prueba de la correcta configuración

Una vez configurados los módulos bluetooth y los de E/S, es conveniente probar su correcto funcionamiento antes de empezar a trabajar con Designer para evitar errores que pueden ser difíciles de encontrar. Lo primero que se puede probar es que todos los módulos bluetooth son accesibles. Para ello, empezando por el punto de acceso, les hacemos un ping y esperamos su respuesta:

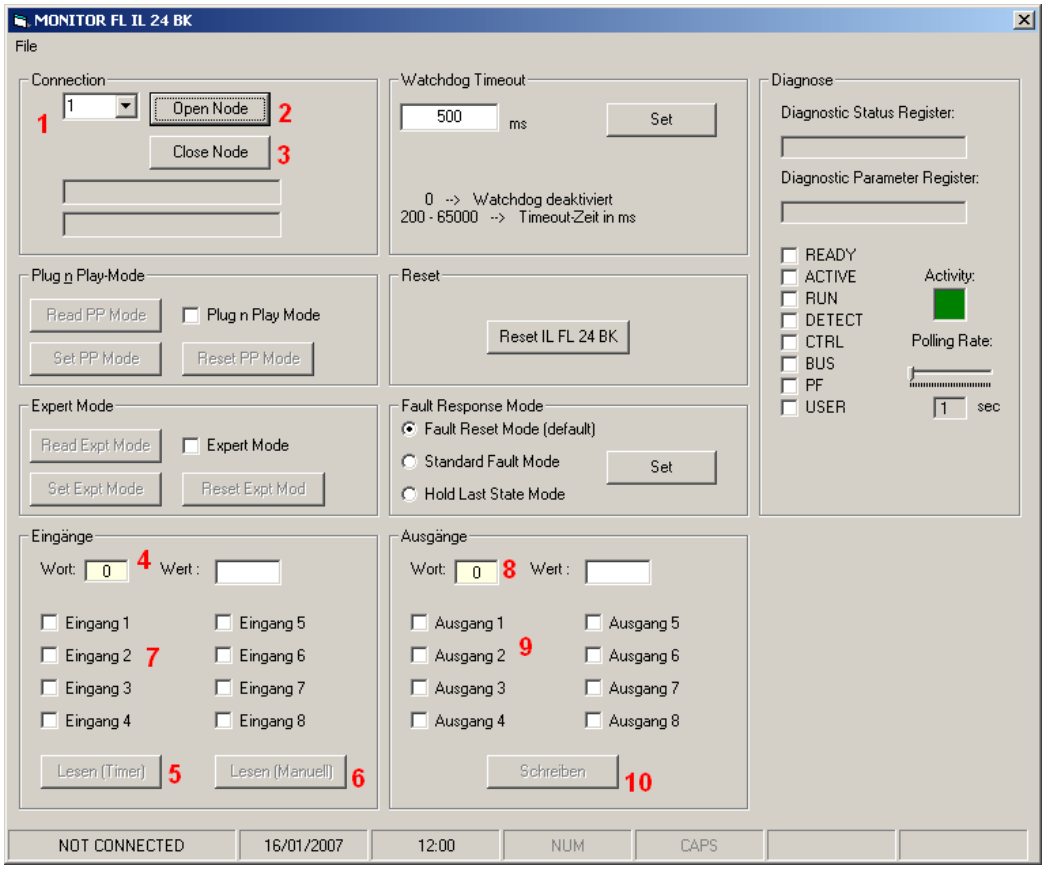


Si el punto de acceso no es accesible, el resto de módulos tampoco lo serán, de manera que se tendrá que revisar la configuración de éste. Lo mas recomendable es revisar en un principio la configuración del propio PC desde el que se está realizando la prueba, por si no esta configurada su red de manera adecuada.

Si el punto de acceso es accesible, pero el resto de módulos no lo son, es conveniente revisar la configuración Bluetooth del punto de acceso, por si hubiese algún dato incorrecto. En caso de no funcionar así, será necesario revisar de manera independiente cada módulo, conectando directamente al PC y revisando su configuración.

Una vez todos los módulos Bluetooth funcionen correctamente, se comprueba, también mediante un ping, que son accesibles los módulos de E/S. En caso de haber errores, será necesario revisar la configuración del módulo bluetooth al que está conectado, en concreto la configuración PPP. Si prosigue el problema, se recomienda volver a configurarlo como se indica en la guía, incluyendo el proceso de asignación de la IP mediante el servidor BootP.

Una vez todos los elementos del sistema sean accesible se puede realizar una prueba del correcto funcionamiento de los módulos de E/S. Para ello se puede utilizar la aplicación "Monitor FL IL 24 BK", ejecutable desde "*FL BK Moni.exe*". Desde esta aplicación se puede seleccionar una estación definida anteriormente con el *commway* y probar que realmente funciona correctamente. Las operaciones importantes a realizar con este programa son las siguientes:



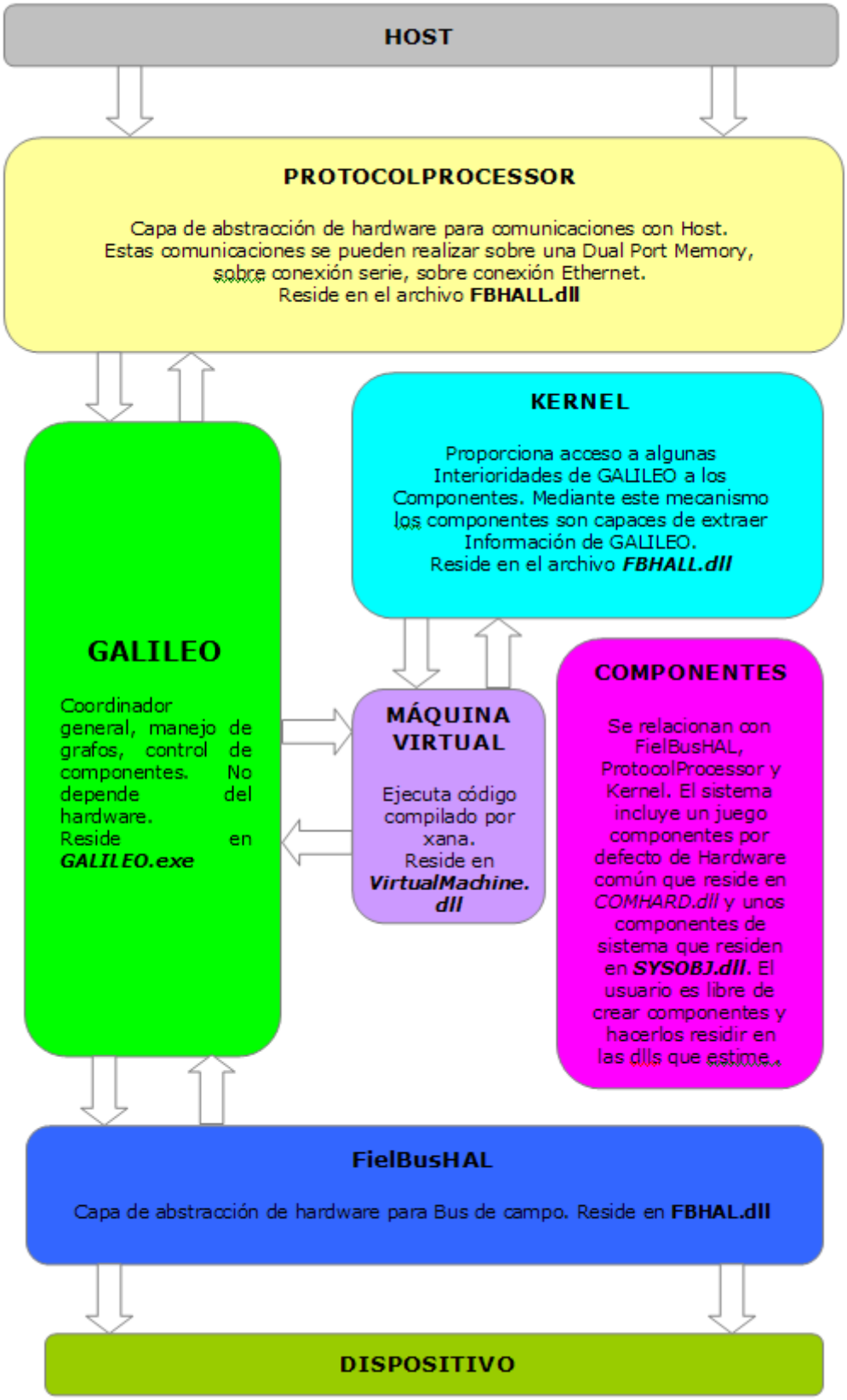
1. Selecciona el módulo al que acceder. Este es el índice que tiene en *commway*.
2. Abre una conexión con el nodo. Las operaciones sobre el nodo requieren que se abra una conexión.
3. Cierra la conexión con el nodo.
4. Selecciona la palabra de entrada que se quiere leer.
5. Lee la palabra de entrada selecciona continuamente.
6. Lee la palabra de entrada seleccionada una sola vez.
7. Marca los bits activos en la palabra leída.
8. Selecciona la palabra de salida que se quiere leer.
9. Bits de la palabra de salida que se quieren escribir.
10. Botón que escribe los bits de la palabra de salida seleccionados.

2 SISTEMA DE CONTROL GALILEO: DEFINICIONES Y ARQUITECTURA

2.1 ARQUITECTURA DEL SISTEMA

La arquitectura del sistema, representada a grandes rasgos, es la siguiente.

SISTEMA DE CONTROL GALILEO



El sistema Galileo se instala como servicio de Windows NT / 2000 / XP / 2003 o bien como ejecutable de nombre GALILEOGUI.EXE en las máquinas Windows9x y Millenium (estas últimas solo para pruebas ya que la aplicación NO ESTÁ CERTICADA EN ESTOS SISTEMAS) y lo hace en los siguientes directorios:

- *C:\GALILEO*: En este directorio reside el ejecutable *galileo.exe* que es el encargado del funcionamiento general, *Thycom.dll* que es la librería de manejo de objetos, y las dll's auxiliares del runtime de C++ (Estas DLL's no se enumeran por que son susceptibles de cambiar con nuevas versiones de los compiladores de C++).
- *C:\GALILEO\THYCOMP*: En este directorio residen los componentes programados por usuario como los componentes nativos del sistema Galileo. En la instalación del Sistema aparecerán VirtualMachine.dll, comhard.dll, sysobj.dll, comcomponents.dll
- *C:\GALILEO\THYCOMP\HAL*: En este directorio residen las capas de abstracción del hardware necesarias para que GALILEO establezca comunicación con las diferentes tarjetas de bus de campo o simulación.
- *C:\GALILEO\CONFIG*: En este directorio se encuentran los archivos de configuración de GALILEO que son:
  - *Galileo.properties*: Archivo encargado de establecer el sistema de Log y configurar los niveles de traza.
  - *Galileo Control System.ini*: Archivo encargado de la configuración de Galileo.
  - *Galileo Mirror System.ini*: Este fichero aparece a partir de la version 3.1, sirve para configurar el sistema de galileos de replica.
  - Ficheros **.prf*: en conjunción con el anterior, sirven para configurar el sistema de replica. No deberían ser modificados por el usuario, sino a través de la configuración de la consola de Galileo.
- *C:\GALILEO\LOGS*: En este directorio se encuentran todos los archivos de Log, así como una utilizada (Wintail) que permite visualizar dinámicamente uno de los archivos de Log. Basta con arrastrar dicho archivo sobre el ejecutable.
- *C:\GALILEO\TEMP*: Este directorio se utiliza por Galileo para acciones temporales. No debe ser borrado.
- *C:\GALILEO\BACKUP*: Este directorio se utiliza por Galileo para preparar las copias de seguridad.

NOTA: *Los directorios en los que se instala Galileo no son opcionales, la razón de esta decisión es por compatibilidad con versiones anteriores.*

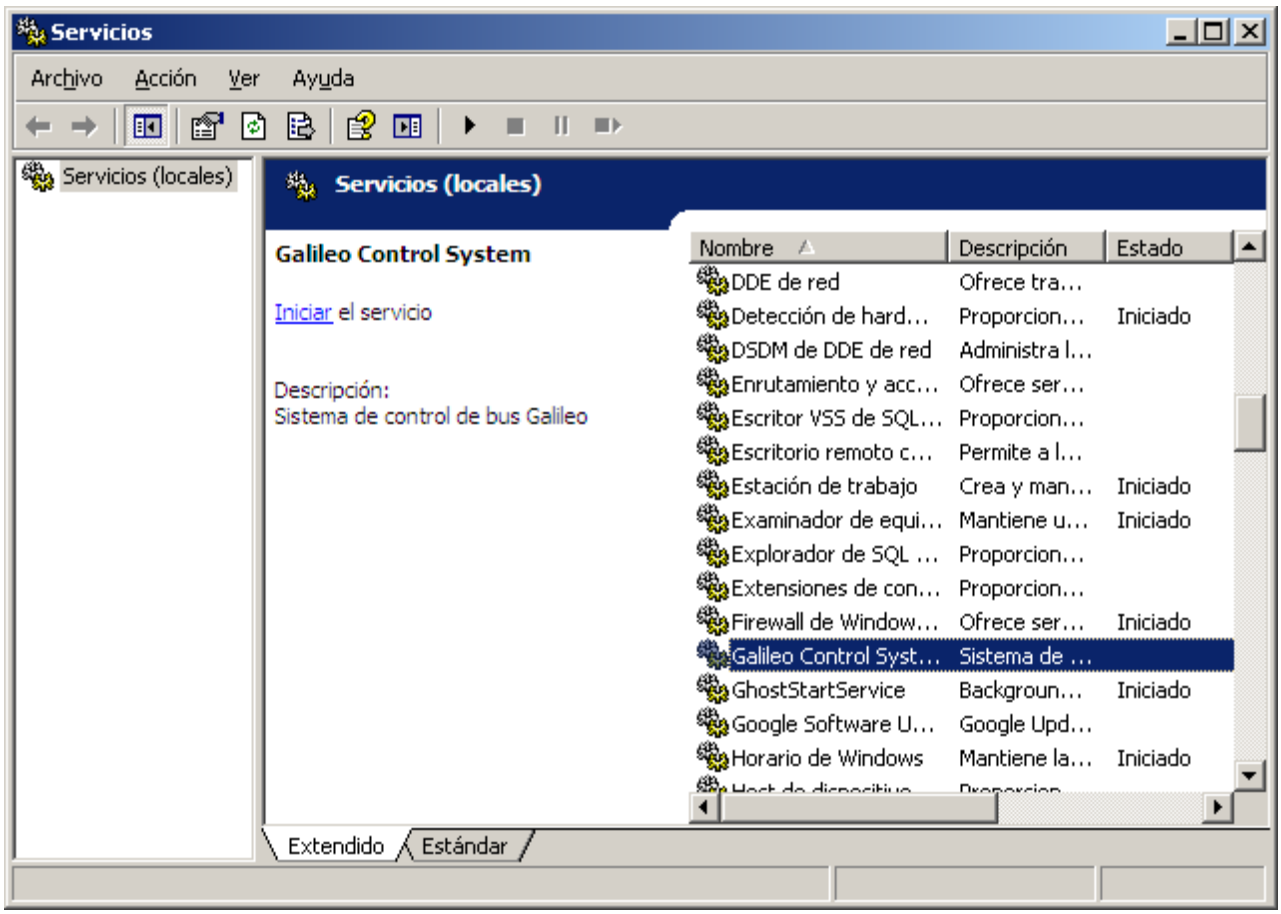
El sistema Galileo está diseñado como un sistema extensible, en el cual es posible añadir librerías de enlace dinámico con nuevos componentes en cualquier momento. Estos nuevos componentes tendrán acceso mediante unos interfaces definidos a estructuras y objetos de utilidad internos de manera que puedan comportarse como los componentes suministrados con el propio sistema. De esta forma se pone a disposición del programador de control una potencia no existente en el mundo de los PLCs, nada que se pueda realizar con un ordenador deja de poder realizarse con Galileo. Esto significa acceso a comunicaciones TCP/IP, acceso a llamadas de procedimientos almacenados en base de datos, envío de E-mail en circunstancias de alarma, etc.
Además del servicio anterior y su arquitectura, la instalación de Galileo también instala un servicio llamado **"***UNISON Syncro server*". Este servicio debe encontrarse en funcionamiento si se desea que el sistema de replica de Galileo funcione correctamente. Debe estar instalado en todos los computadores que forman parte de la red Galileo, y al igual que Galileo, también necesita que las comunicaciones por TCP/IP funcionen de forma correcta, y que el puerto de comunicaciones 13032 este habilitado (es decir, no restringido por un Firewall u otra aplicación de gestión de red). Si el puerto por defecto de Galileo para comunicaciones se cambiara, también sería necesario cambiar este servicio, puesto que su número debe ser el del puerto de comunicaciones de Galileo +10.
Para hacer este cambio, es necesario modificar el registro de windows, buscando la clave:
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\UNISON Syncro server\Parameters
Y dentro de ella reemplazar el valor del parámetro "Application" por algo del estilo:
C:\Galileo\Unison.exe -socket XXXX
Donde XXX es el número de puerto nuevo (= puerto de comunicaciones de Galileo + 10).

2.2 COMO EJECUTA GALILEO LAS APLICACIONES

2.2.1 Conceptos generales del sistema de ejecución

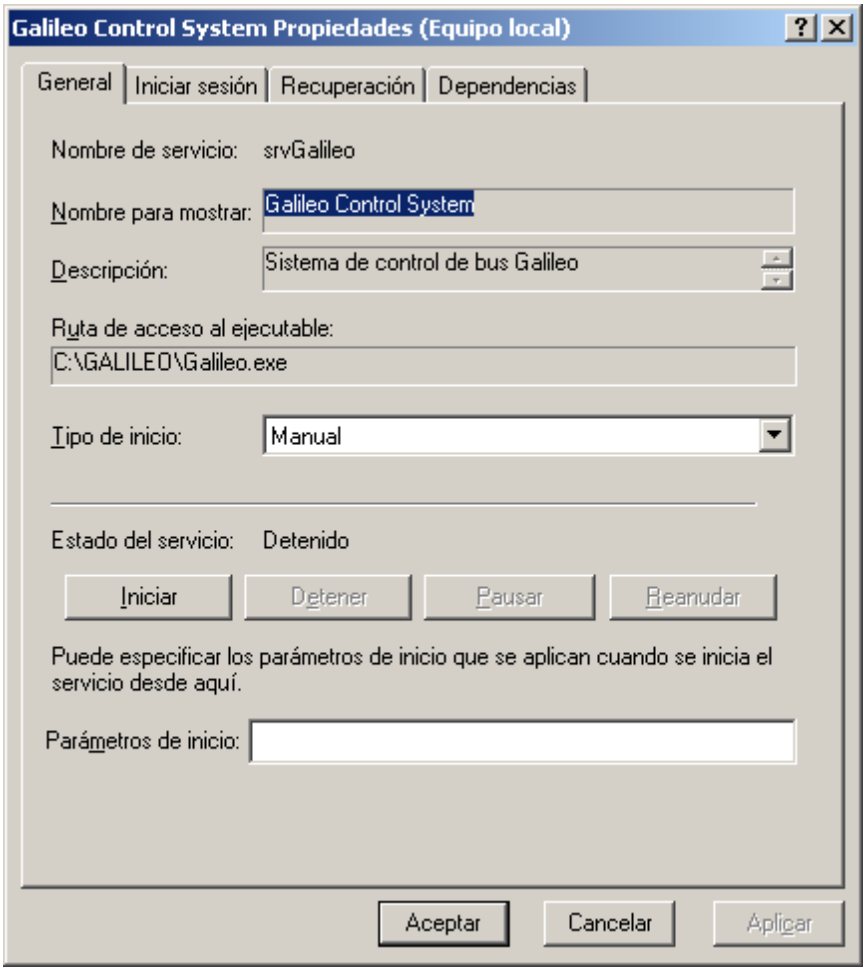
El programa de control será ejecutado en un PC destinado a esta tarea. En este PC debe estar instalada una tarjeta de comunicaciones de bus de campo con la cual poder realizar tareas de control. Este PC, debido a las características de funcionamiento del sistema, debe ser un PC rápido (>800 Mhz) con memoria (>256 Mbytes) y equilibrado (el acceso a la red o disco duro no debe "ahogar" el funcionamiento del programa de control ni quitar estabilidad a su tiempo de ciclo).

Una vez instalado el software de control, se apreciará que aparece un servicio nuevo instalado en el applet de control de servicios del panel de control.



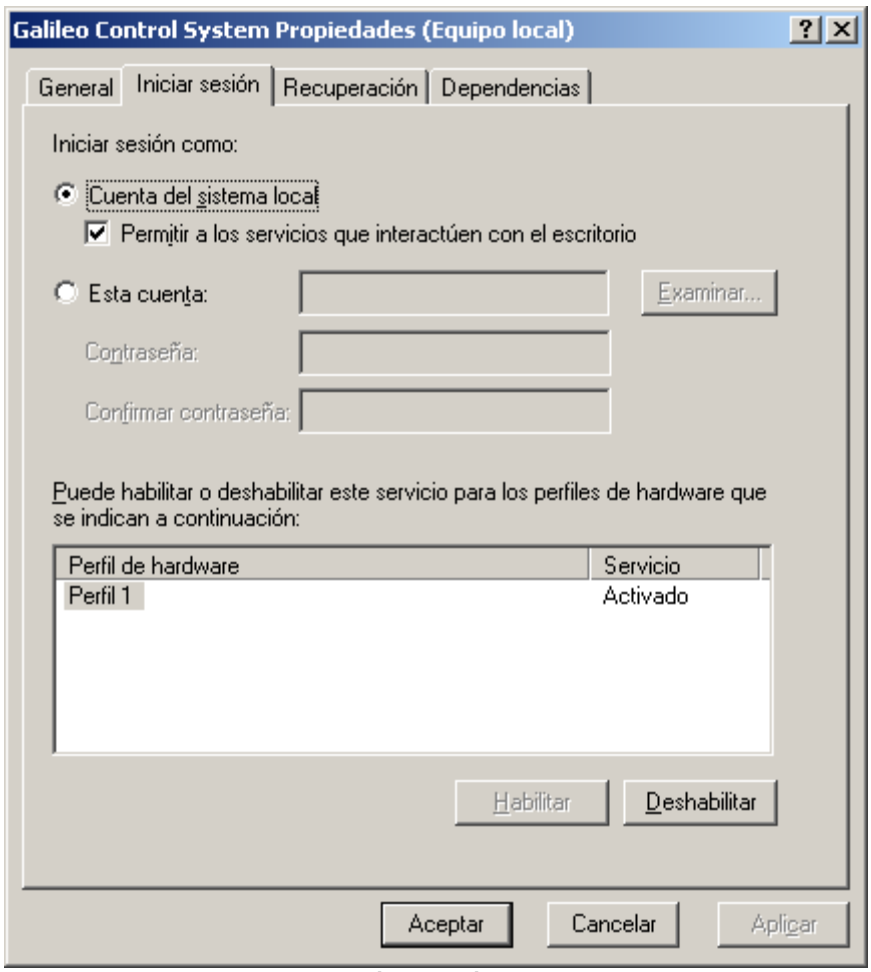
Galileo Control System es un servicio de Windows que por defecto se instala para una ejecución automática. Esto significa que el PC de control al arrancar y sin que nadie realice Logon en la máquina comenzará la ejecución de la aplicación. Lógicamente este es el comportamiento deseado, aunque en determinadas ocasiones (sobre todo en las pruebas al inicio de la puesta en marcha) puede que no interese este modo de funcionamiento, para lo cual bastará con colocar el servicio en manual.

El SCM (Service Control Manager) de Windows nos permite en todo momento cambiar y visualizar el modo de funcionamiento del servicio.



Este servicio se instala a su vez bajo credenciales de *LocalSystem* (*LocalSystem* es un usuario con todos los privilegios posibles) y en modo interactivo, lo que lógicamente significa que puede interactuar con el usuario. Esta interactividad se utiliza para cambiar parámetros de su configuración, visualizar tiempos de ciclo, activar y desactivar trazas, etc.

Estas opciones no deben ser cambiadas sin consultar previamente con el departamento de soporte.



Todo lo anteriormente descrito también es válido para el otro servicio que se instala con la versión 3.1: *UNISON Syncro Server*, que se encargar de proporcionar comunicaciones al sistema de réplica de Galileo en esa versión.

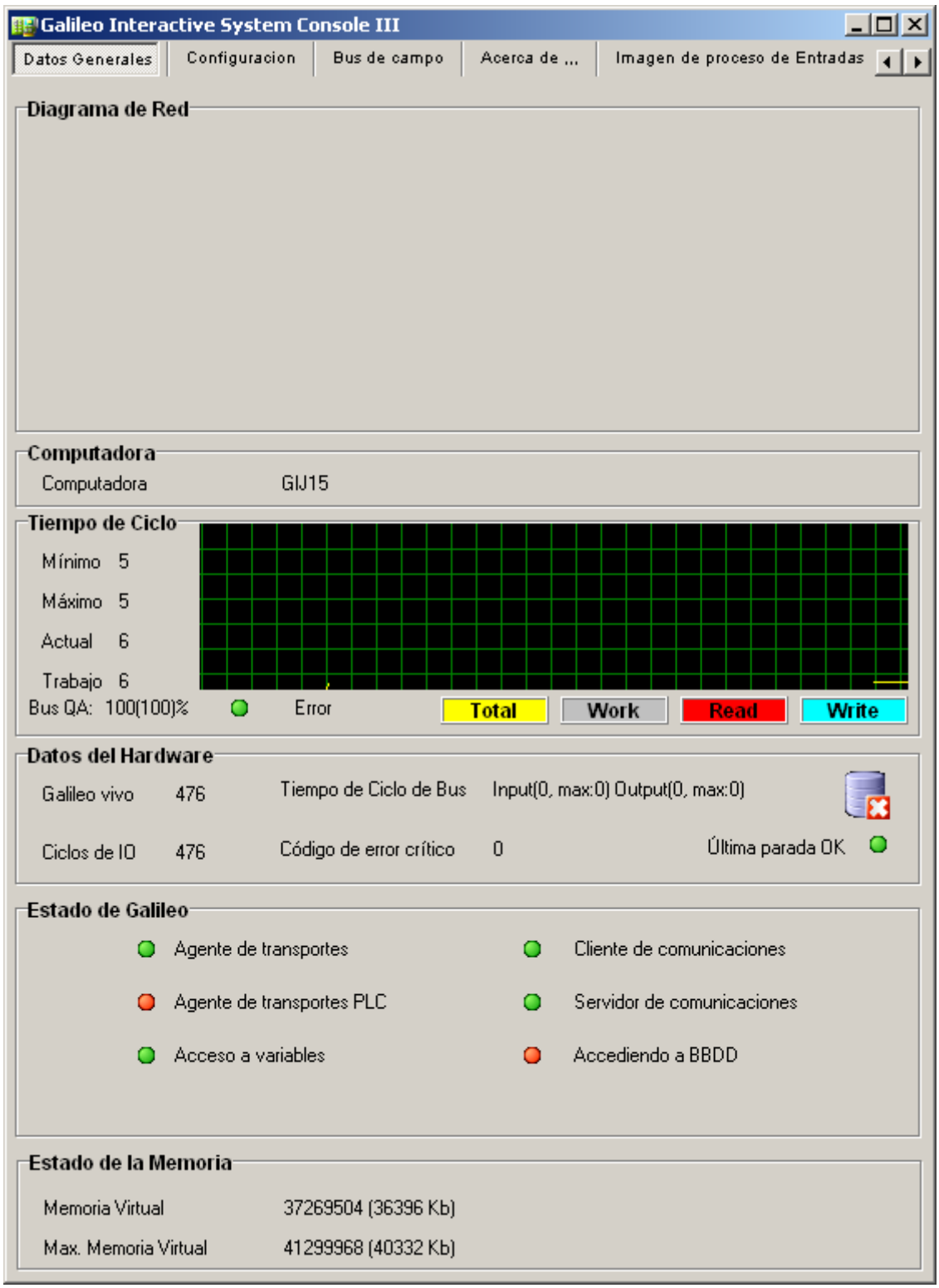
2.2.2 La consola de Galileo (configuración del sistema)

Al ser un servicio, Galileo no dispone de un interfaz directo de manejo. Para poder interactuar con él, es necesario ejecutar la aplicación "*GalileoConsole.exe*" que se instala junto con el resto de Galileo. Esta aplicación no se inicia automáticamente cuando se inicia Galileo, y puede ser detenida de forma independiente. En caso de que se detenga es posible iniciarla manualmente sin necesidad de reiniciar el sistema GALILEO. Una vez arrancada esta aplicación, aparecerá la pantalla principal que nos deja visualizar algunas informaciones importantes.
Esta aplicación también aparecerá minimizada en la barra del Sistema Operativo y será necesario pulsar doble clic sobre ella, o bien botón derecho para mostrar el menú contextual donde podremos seleccionar Mostrar.

En la barra la consola aparecerá como un microchip, en diferentes posiciones en función de si se está o no ejecutando el servicio de Galileo.
Una vez que pulsamos mostrar, aparecerá el interface gráfico de dicha aplicación en la cual las solapas principales son las siguientes:

2.2.2.1 Datos Generales

La primera solapa de la aplicación muestra la pantalla de **Datos generales**:

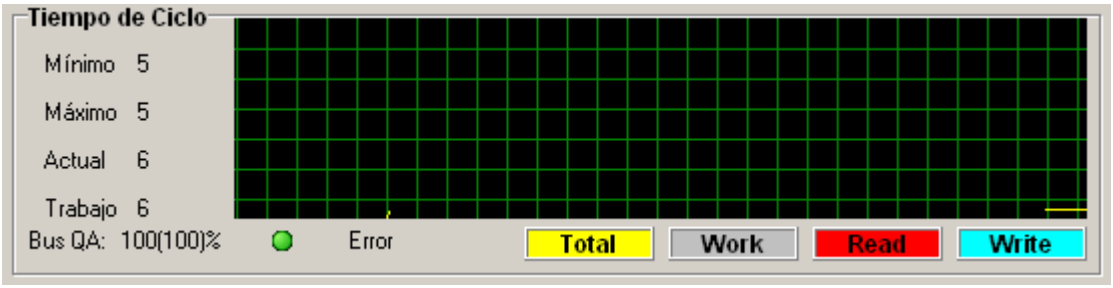


En ella podemos visualizar los siguientes datos:

- Diagrama de Red: Muestra las computadoras del proyecto. GALILEO realiza una comunicación cíclica entre computadoras de un mismo proyecto para adquirir la imagen de proceso de las mismas. Dichas computadoras aparecen en este diagrama. Los iconos representados son iconos con animación que mostrarán dos computadores realizando transferencia de datos si la comunicación entre el computador donde se ejecuta la consola y el computador representado es correcta, si no lo es presentará una diana para indicar el punto de fallo. El Hint de estos controles indicará la fecha y hora de la última comunicación. ***El fallo solamente se representa a partir de 30 segundos sin comunicaciones con el computador.***

- Computadora: Indica la estación en la cual se está ejecutando la aplicación

- Tiempo de ciclo:



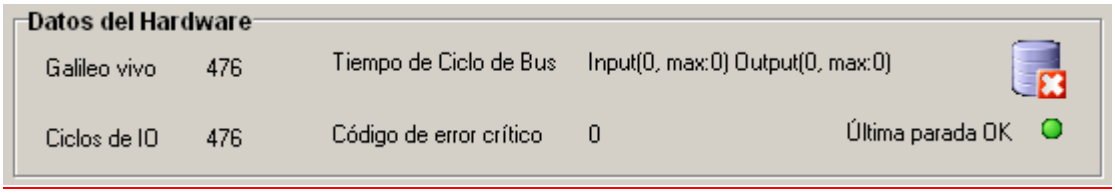
Además de estos datos aparecen datos relativos al tiempo de ciclo de programa. En esta ventana se muestran los siguientes valores:

- ✓ Mínimo**:** tiempo de ciclo de mínimo.
- ✓ Máximo: tiempo de ciclo máximo.
- ✓ Actual**:** último tiempo de ciclo. Como tiempo de ciclo se toma el tiempo de lectura de entradas, ejecución de código, escritura de salidas y retardo del ciclo.

- ✓ Trabajo**:** tipo de ciclo sin contar el correspondiente a la lectura/escritura de entradas/salidas.
- ✓ Bus QA**:** calidad del bus. En este punto se muestra la calidad actual (calculada en función de las ultimas transmisiones) y entre paréntesis la calidad global, calculada en función de todas las transmisiones realizadas.

Además, se muestra un gráfico continuo a modo de traza del tiempo de ciclo. En este gráfico podremos visualizar si hay picos en el tiempo de ciclo de ejecución. Por último, en este mismo apartado se muestra un LED que indica si existe algún problema en el bus. Cuando este LED aparece en verde, el bus está funcionando sin problemas, cuando aparece en rojo significa que existe algún problema, *que no tiene necesariamente que producir una caída de bus***.** Podría ser, por ejemplo, un problema en algún dispositivo.

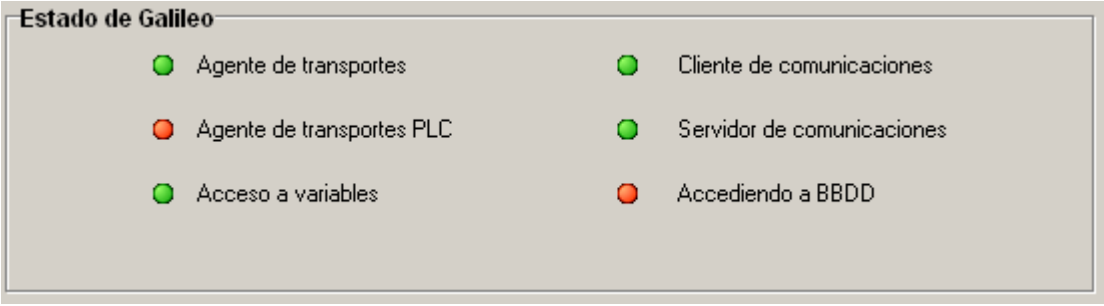
- Datos de hardware:



En el apartado de datos de hardware aparecen datos interesantes tanto para el departamento de soporte como para el programador.

- ✓ Galileo Vivo: Es un contador que incrementa el sistema Galileo en cada vuelta de ejecución (este iniciado o no el programa de usuario).
- ✓ Ciclos de IO: Indica la cantidad de ciclos de intercambio de entradas salidas efectuados con éxito.
- ✓ Tiempo de ciclo de bus: Indica el tiempo de ciclo de bus actual. Este es un cálculo estimativo y puede ser perfectamente valor 0.
- ✓ Código de Error Crítico: Indicaría el código de error en última instancia antes de que Galileo interrumpa la ejecución del programa de usuario (en caso de ser distinto de cero).
- ✓ LED Última parada OK: Indica si la última parada que ha realizado el servicio Galileo ha sido controlada por el usuario (LED en verde). En caso contrario, es decir, con el LED en rojo, indicaría que la última parada que se ha producido ha sido por motivos desconocidos.

- Estado de Galileo:



Además, aparece también información referente a los subsistemas de Galileo, y a su estado de ejecución. En concreto, se muestra el estado de los agentes de transportes y de los sistemas de comunicaciones. Estos sistemas, en condiciones normales tienen que estar en verde (o al menos, ponerse en verde cada cierto tiempo), a excepción del sistema del agente de transportes PLC que puede estar apagado si no existen PLCs definidos.

Junto con estos datos se muestra información acerca de la ejecución del programa. En concreto, existen dos LEDs que indican ciertas circunstancias especiales durante la ejecución:

- ✓ LED "acceso a variables": Este LED estará en verde mientras el programa de control no intente acceder a variables inexistentes. Si en algún momento el programa de control intenta acceder a una variable inexistente (o declarada como privada) pueden ocurrir dos cosas:

  → Galileo detiene la ejecución informado del error. Esto se producirá si en la pestaña de configuración está activa la opción "Checkear acceso a variables".

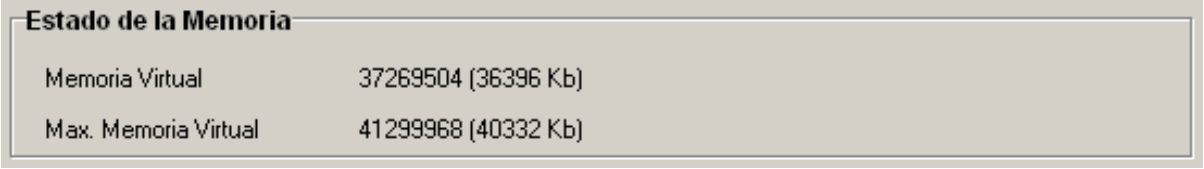
  → Galileo no detiene su ejecución y continúa (como en versiones anteriores) pero pone a rojo este LED y además escribe un mensaje de error en el log Galileo.log. Esto se produce si en la pestaña de configuración no está activa la opción checkear acceso a variables.

- ✓ LED "accediendo a BBDD": Este LED estará en verde cuando Galileo esté accediendo a cualquiera de las funciones de la base de datos desde el agente de transportes. Por regla general este LED se mostrará en rojo, ya que estás operaciones suelen ser muy rápidas. A pesar de esto, es conveniente revisar este LED cuando se sospeche que las operaciones en base de datos son lentas. En este caso, el síntoma que indica que existe un problema de lentitud en la base de datos (u otros problemas que hacen que el tiempo que se consume en esas funciones es alto), es que este LED se mantiene en verde. Si además de este LED en verde, está el LED del agente de transportes en rojo (o el del agente de transporte PLC en caso de existir), existen altas posibilidades de que el agente de transportes esté parado dentro de alguna de las funciones de la base de datos.

- ✓ LED Agente de Transportes: Indica si existe comunicación con el Agente de Transportes en el caso de que el programa de control esté realizado con Galileo íntegramente.

- ✓ LED Agente de Transportes PLC: Indica si existe comunicación con el Agente d Transportes en el caso de que el programa de control esté realizado con PLC comercial.
- ✓ LED Cliente de comunicaciones: Indica si existe comunicación con el resto de clientes de los Galileos con los cuales está trabajando como servidor.
- ✓ LED Servidor de comunicaciones. Indica si existe comunicación con el resto de servidores de de los Galileos de los cuales es cliente.

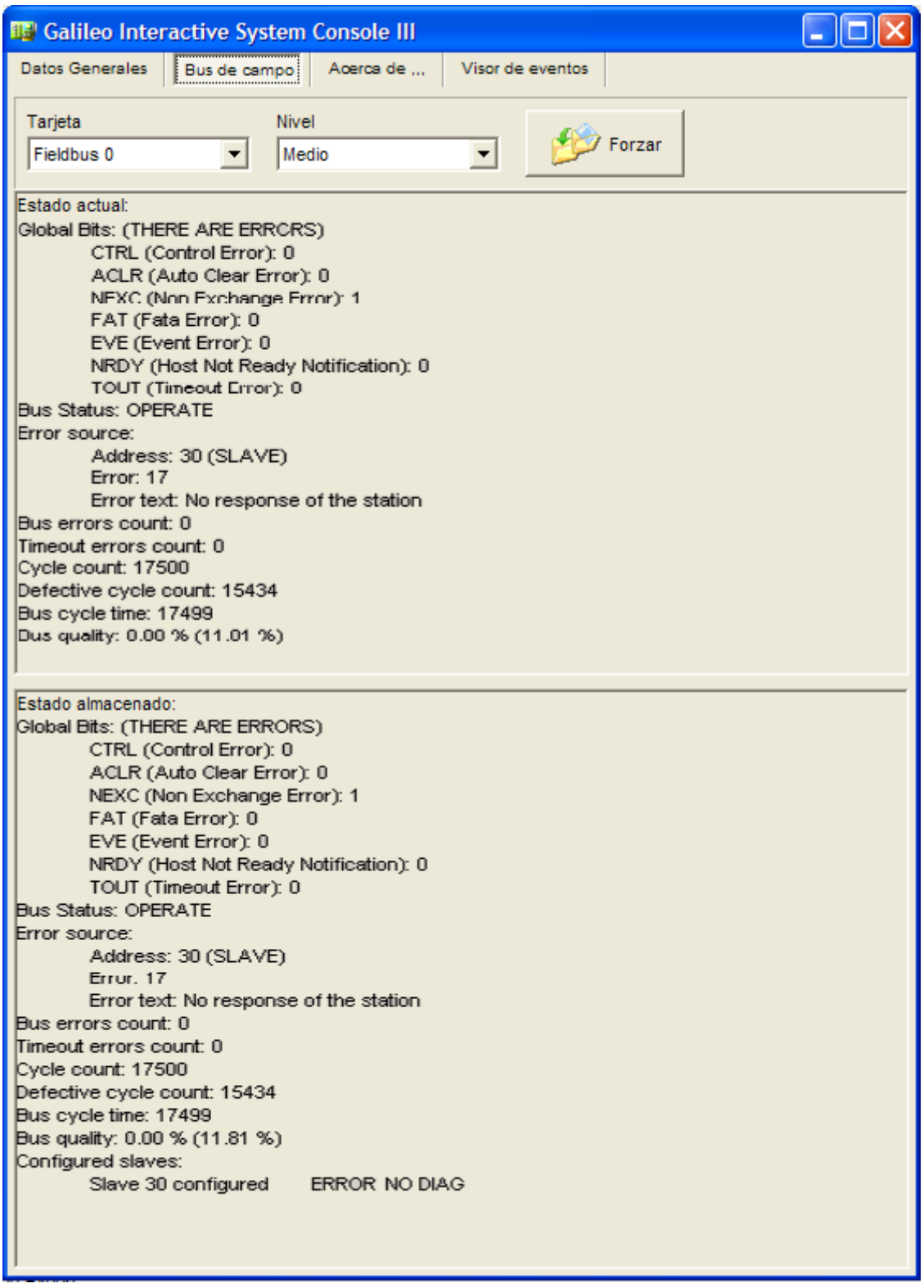
- *Estado de la memoria*:



Por último, aparece información acerca de la memoria consumida por Galileo. Estos valores se actualizan constantemente y únicamente son una indicación para conocer el consumo de memoria.

2.2.2.2 Bus de campo

Desde esta pestaña se puede hacer un diagnostico en profundidad del estado del bus de campo. La información aquí mostrada depende del tipo de tarjeta y del fabricante, por lo que existen casos en los que no se mostrarán errores.



Esta pantalla se divide en dos partes:

- *Superior*, se muestran los errores actuales en el bus

- *Inferior*, se muestra el último error de bus ocurrido.

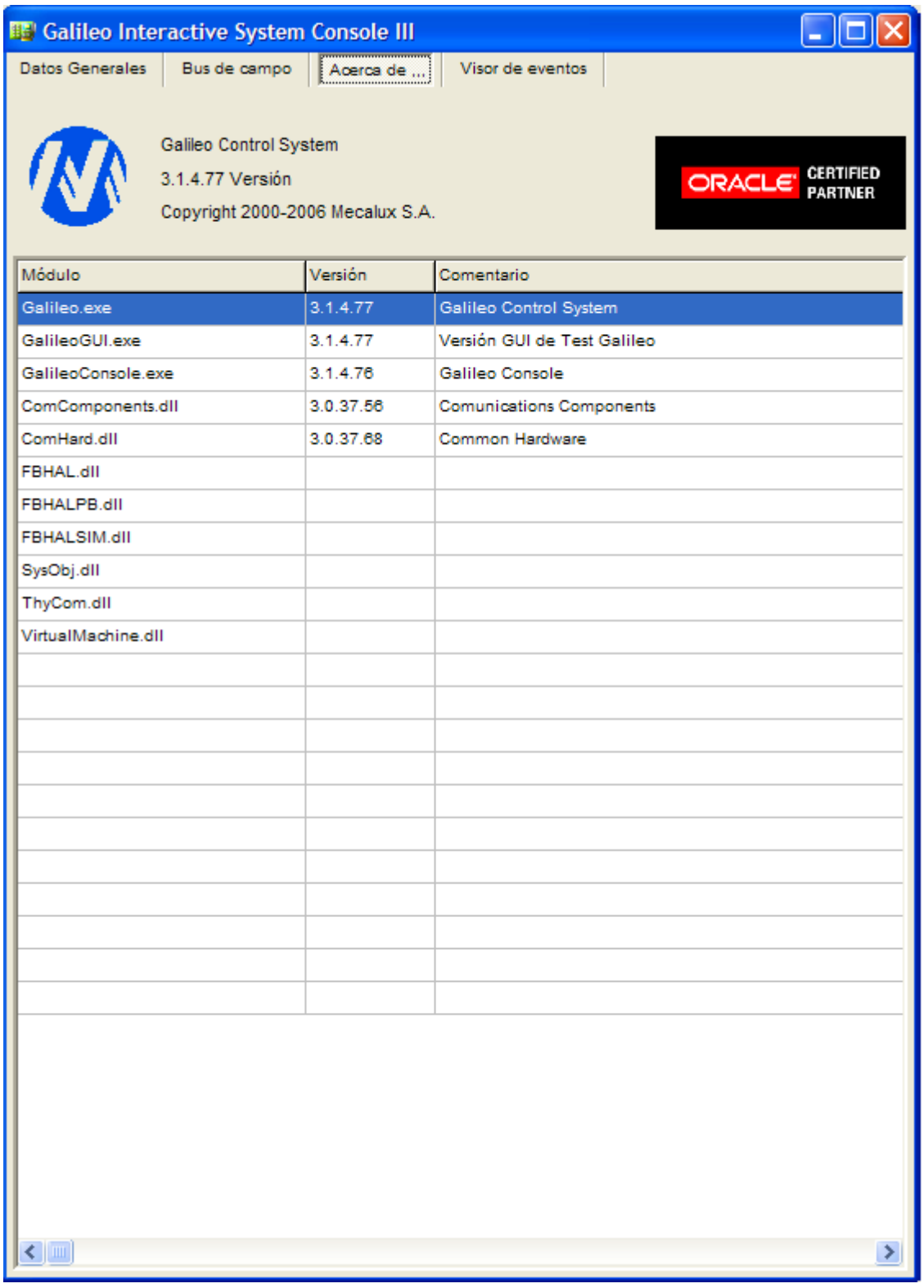
Existe además, dos selectores que permiten elegir la tarjeta a monitorizar y el nivel de diagnostico. Como se dijo antes, este diagnóstico es dependiente del hardware por lo que los datos mostrados en cada nivel son diferentes. De todas maneras, los siguientes datos son válidos para todo el hardware disponible:

- Nivel bajo: no muestra datos, y solo actualiza la señal de error que se muestra en la pestaña general.
- Nivel medio: muestra datos de diagnóstico que tengan poca incidencia en el tiempo de ciclo. En el caso de Profibus, estos datos se consiguen durante el propio intercambio de E/S, por lo que el tiempo de ciclo se ve muy poco afectado.
- Nivel alto: muestra todos los datos anteriores, pero además, escribe datos de diagnóstico más profundos en el log FBDiagnostics.log. *La obtención de estos datos puede tener influencia en el tiempo de ciclo, por lo que tiene que utilizarse con sumo cuidado.*

Para activar un nuevo nivel de diagnostico es necesario seleccionarlo y a continuación pulsar el botón Forzar para que se active.
Para el caso de Profibus, el nivel Medio es más que adecuado para un diagnóstico completo. Con este nivel se muestra el estado de todos los bits de error (Global bits) de manera que cuando uno de ellos esté a uno significa que existe algún problema. Se muestra el estado general del bus (operate, offline, etc) y estadísticas de ciclos de error y ciclos correctos. Por último se muestra la lista de esclavos configurados y el estado de cada uno de ellos.

2.2.2.3 Acerca de …

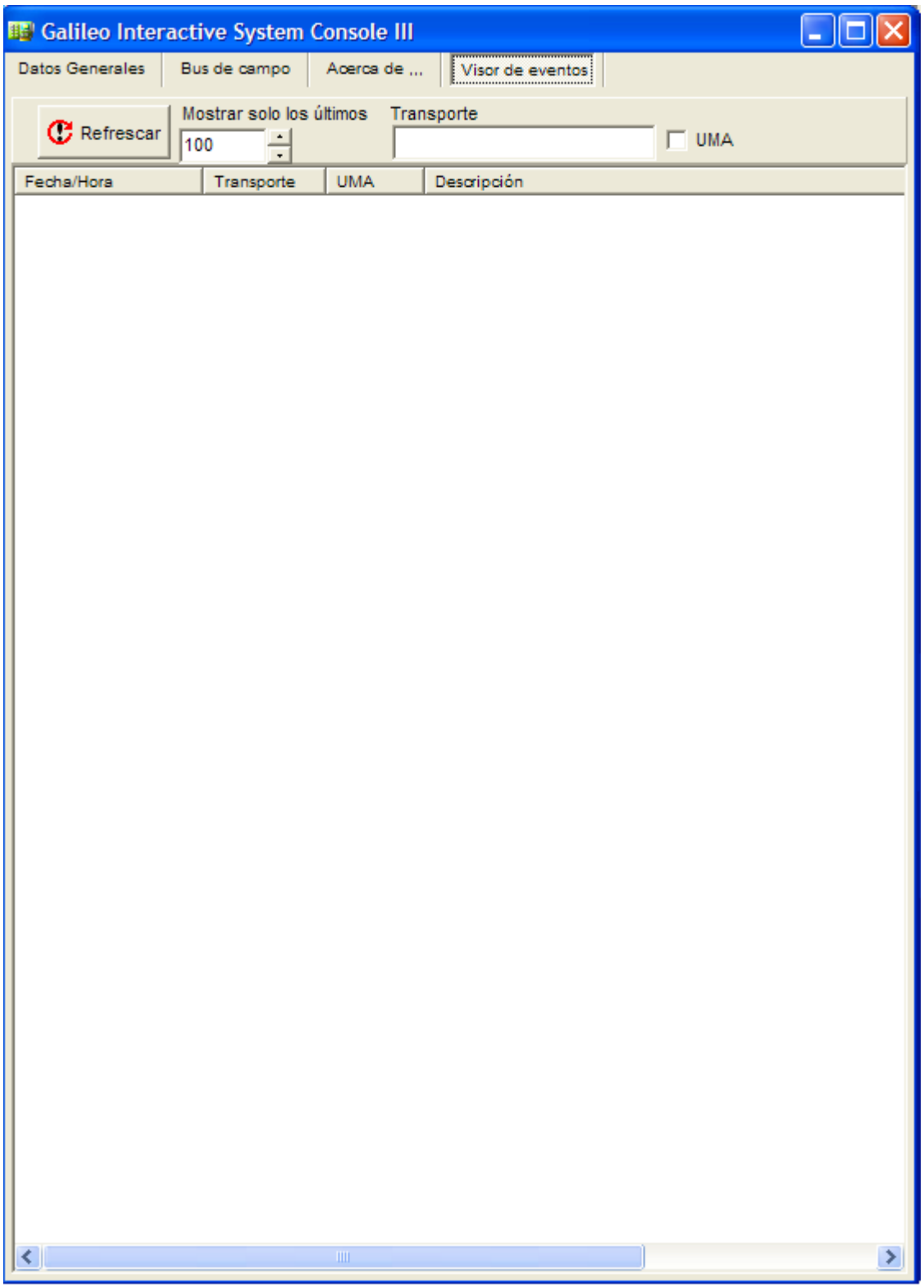
Además de estas solapas aparece otra donde se muestra la información de la aplicación. Esta información debe ser indicada siempre ante una llamada al departamento de soporte



En esta pantalla se nos muestra la versión general de la aplicación en la parte superior y en la tabla que se encuentra en la parte inferior se muestra la versión de cada uno de los componentes del sistema.

2.2.2.4 Visor de Eventos

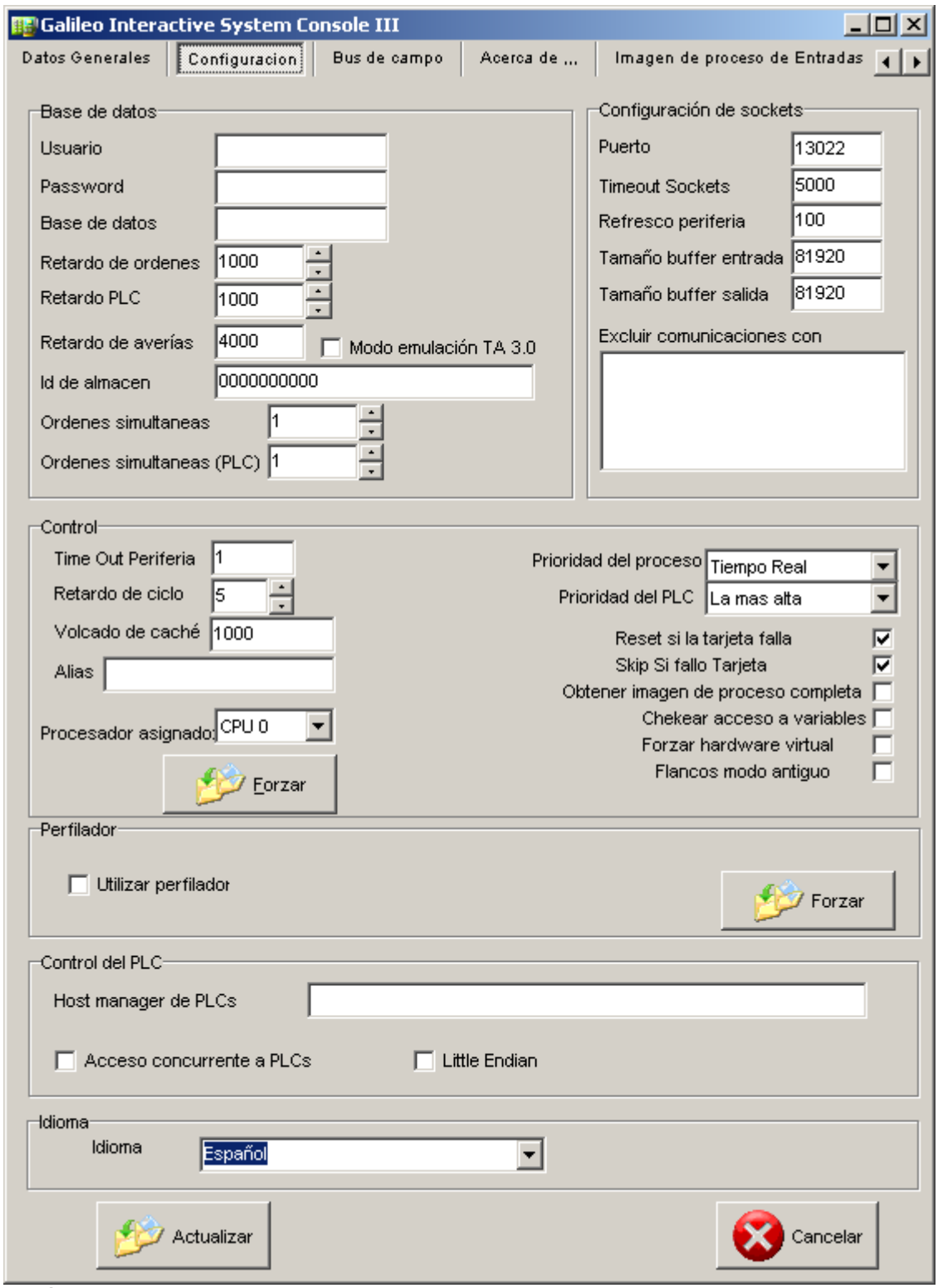
La última pantalla de las inicialmente visibles muestra información recogida en la base de datos sobre los eventos ocurridos en transporte últimamente.



Es posible realizar consultas sobre dicha pantalla para filtrar por número de transporte o unidad de transporte del almacén (Palet, bandeja, etc.).
Pulsando sobre el botón "Refrescar" se actualizará dicha información con los últimos cambios registrados.

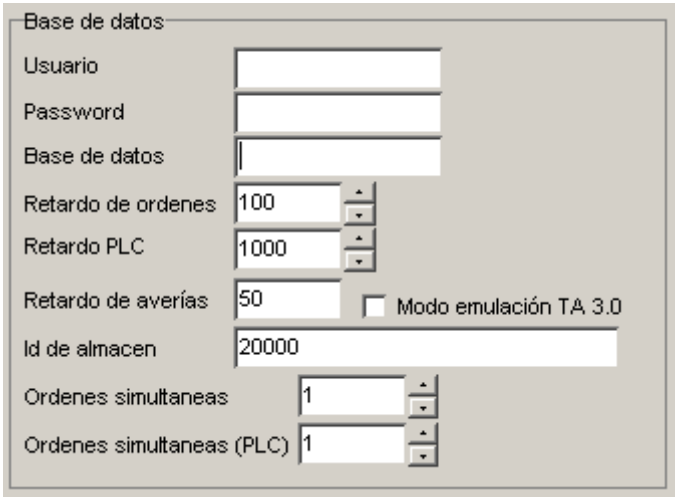
Como cualquier usuario tiene acceso a esta aplicación, se ha establecido un método oculto para configurar el sistema. Este método consiste en escribir la palabra ***thyman*** sobre el cuadro de diálogo. En ese momento aparecerán más solapas que nos permitirán realizar una serie de acciones adicionales.

2.2.2.5 Configuración de parámetros



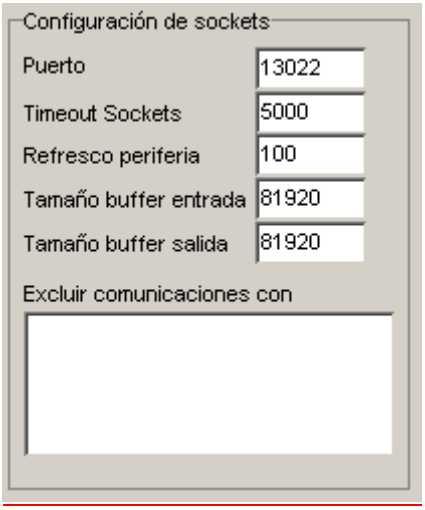
Aquí podemos (y debemos) establecer los siguientes valores:

- Base de datos:



1. *Usuario*: Éste será el usuario mediante el cual el agente de transportes se conectará a la base de datos.
2. *Password*: Éste será el password mediante el cual el agente de transportes se conectará a la base de datos.
3. *Base de Datos*: Éste será el alias de Oracle al cual nos deseamos conectar, corresponde a un nombre de la configuración TNS de Oracle.
4. *Retardo Ordenes*: Es el tiempo en que se duerme el proceso Agente de Transportes entre una búsqueda de ordenes y otra.
5. *Retardo PLC*: Es el mismo tiempo que en el caso anterior pero referenciado a aquellas instalaciones en las cuales esté realizado el control mediante PLC comercial.
6. *Retardo Averías*: Es el tiempo en que se duerme el proceso de log de averías a base de datos entre un procesamiento y otro.
7. *Modo emulación TA 3.0*: indica a Galileo que se ejecute utilizando el agente de transportes 3.0
8. *Id de almacén*: para las instalaciones que funcionen con el modo nativo de 3.1, se tiene que indicar el identificador de almacén con el que se conectará a la base de datos y ha de ser proporcionado siempre por el programador del Sistema de Gestión.
9. *Órdenes simultáneas*: Indica el número de funciones del Agente de Transportes que se van a ejecutar simultáneamente (búsquedas, eventos, etc.). El funcionamiento existente antiguamente es el que aparece por defecto con valor 1. Ha de ser ajustado correctamente en cada instalación
10. *órdenes simultáneas (PLC)*: Indica el número de funciones del Agente de Transportes PLC que se van a ejecutar simultáneamente (búsquedas, eventos, etc.). El funcionamiento existente antiguamente es el que aparece por defecto con valor 1. Ha de ser ajustado correctamente en cada instalación.

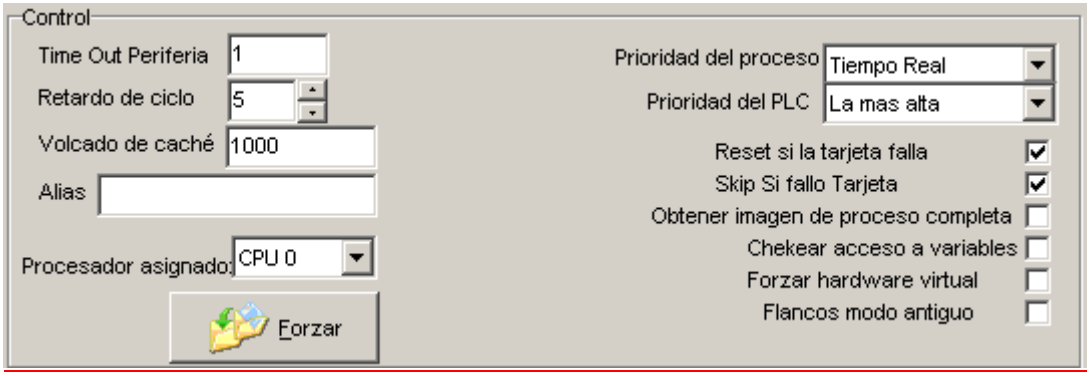
- Configuración de sockets:



1. *Puerto*: En la configuración de sockets aquí se configura el puerto, ***debe ser 13022***, no se aconseja modificar esta opción sólo sin consultar con el departamento de soporte.
2. *Timeout de Sockets*: Tiempo permitido para establecer las conexiones de red, **debe ser 5000**, esta opción solo puede ser modificada por el departamento de soporte.
3. *Refresco Periferia*: Este tiempo es el intentará mantener GALILEO para actualizar la periferia de las demás computadoras del proyecto. En todo caso es estimativo y depende de las condiciones del entorno (Red, tipo de cableado, trafico en el medio físico) para poder cumplir el objetivo marcado.
4. *Tamaño buffer entrada*: Define la cantidad de bytes que se van a recibir en cada paquete de datos en la comunicación entre PCs.
5. *Tamaño buffer salida*: Define la cantidad de bytes que se van a enviar en cada paquete de datos en la comunicación entre PCs.

   Estos dos últimos parámetros de configuración no es necesario ajustarlos si la comunicación entre PCs tiene tiempos de transmisión normales (valor por defecto 8Kb).
6. *Excluir comunicaciones*: desde esta opción se pueden añadir computadores (haciendo clic con el botón derecho en la lista) que serán excluidos de las comunicaciones de Galileo. La exclusión se refiere únicamente a que Galileo *NO PEDIRA* datos a estas máquinas, aunque siempre responderá a peticiones suyas. De esta manera se puede minimizar la carga de red.

- Control:



1. *Timeout Periferia*: Dado que las comunicaciones se realizan con el driver en modo interrupción, este timeout sirve para abandonar la espera de datos nuevos al intentar adquirirlos de la periferia. Típicamente el tiempo necesario que necesita el dispositivo para realizar el intercambio con el bus es de 0.42 ms y dado que se hacen dos ciclos de intercambio por ciclo de programa el valor recomendado es 1 ó 2. **NOTA**: *Se han observado tendencias erráticas en el driver de Hilscher con tiempos mayores de 10, llegando a producir la caída del sistema operativo y apareciendo la típica pantalla azul mediante la cual se inicia el volcado físico a disco.*
2. *Retardo de ciclo:* Tiempo en mseg. que se espera entre ciclo y ciclo de ejecución de Galileo.
3. *Obtener imagen de proceso completa*: marcando esta opción, Galileo pedirá al resto de Galileos su imagen de proceso completa, mientras que si no se marca, solo se pedirán las variables públicas. Por defecto, esta opción tiene que estar sin marcar, siendo útil *únicamente* cuando el proyecto que se está ejecutando tiene un número muy grande de variables públicas (mas de 4000 variables públicas).
4. *Reset si la tarjeta falla*: esta opción indica a Galileo que ante un fallo en la tarjeta debe poner a cero todas las entradas. El comportamiento exacto de esta opción depende también de la opción "*Skip si fallo de tarjeta*"
5. *Skip si fallo de tarjeta*: esta opción indica a Galileo como comportarse ante el fallo de alguna de las tarjetas cuando otras si funcionan correctamente. Cuando esta opción se marca, se indica a Galileo que tiene que proseguir la ejecución únicamente con las tarjetas que funcionan, mientras que cuando esta sin marcar Galileo asumirá que todas las tarjetas han fallado. Esta opción tiene importancia en el comportamiento de la opción "*Reset si la tarjeta falla*". Cuando no esta activa, y se produce un error en una de las tarjetas, aunque el resto funcionen correctamente, se asume que todas han fallado, por lo que se ponen a cero todas las entradas si la opción "*Reset si la tarjeta falla*" esta también activa. En la siguiente tabla se muestran todas las combinaciones posibles:

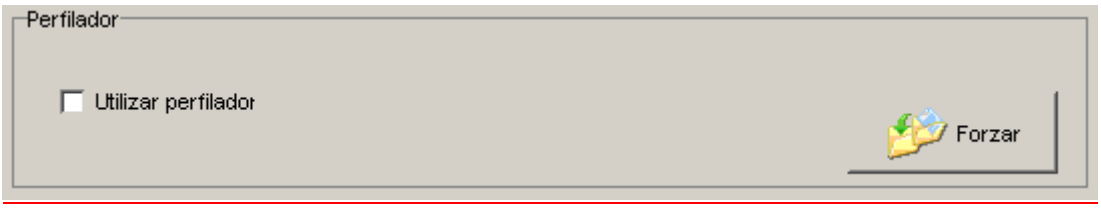
Reset	Skip	Estado tarjetas	Entradas	Salidas	Ejecución
0	0	Todas correctas	Lo leído de las tarjetas	Lo escrito	SI
0	0	Todas con fallo	Lo leído de las tarjetas	Se ponen a cero las salidas de la todas las tarjetas	NO
0	0	Algunas funcionan otras no	Lo leído de las tarjetas	Se ponen a cero las salidas de todas las tarjetas	NO
1	0	Todas correctas	Lo leído de las tarjetas	Lo escrito	SI
1	0	Todas con fallo	Se ponen a cero todas las entradas	Se ponen a cero todas las salidas de todas las tarjetas	NO
1	0	Algunas funcionan otras no	Se ponen a cero todas las entradas	Se ponen a cero las salidas de todas las tarjetas	NO
0	1	Todas correctas	Lo leído de las tarjetas	Lo escrito	SI
0	1	Todas con fallo	Lo leído de la tarjeta	Se ponen a cero las salidas de la todas las tarjetas	SI
0	1	Algunas funcionan otras no	Lo leído de las tarjetas	Se ponen a cero las salidas de las tarjetas que han fallado	SI
1	1	Todas correctas	Lo leído de las tarjetas	Lo escrito	SI
1	1	Todas con fallo	Se ponen a cero todas las entradas	Se ponen a cero las salidas de todas las tarjetas	SI
1	1	Algunas funcionan otras no	Se ponen a cero las entradas de las tarjetas que han fallado	Se ponen a cero las salidas de las tarjetas que han fallado	SI

6. *Checkear acceso a variables*: marcando esta opción forzamos a Galileo a que detenga su ejecución cuando el programa de control intente acceder a una variable inexistente o declarada como privada. El acceso a variables inexistente mediante operaciones `GetVarValue` o `SetVarValue` no es una buena práctica, aunque estaba permitida y era utilizada en versiones anteriores. Desactivando esta opción Galileo no se detendrá cuando encuentre un acceso incorrecto, pero pondrá en rojo el led "Acceso a variables" de la pestaña general y mostrará un mensaje en archivo Galileo.log.
7. *Forzar hardware virtual*: marcando esta opción, forzamos que todas las tarjetas configuradas en este computador utilicen el hardware

virtual, en lugar del hardware físico configurado. Esto evita tener que recompilar el proyecto para realizar ciertas pruebas.

8. *Retardo Ciclo*: Dado que el sistema ejecuta cíclicamente la aplicación en una prioridad muy alta, se deben dar pausas al mismo para que el sistema operativo sea capaz de repartir tiempo de CPU al resto de procesos. Este tiempo representa el tiempo que se duerme el proceso antes de ejecutar un nuevo ciclo. Lógicamente, si se incrementa este tiempo, se incrementa también el tiempo de ciclo. El ajuste se debe realizar en función del tiempo de ciclo deseado y el grado de CPU utilizado por el proceso. (Sería correcto un tiempo de ciclo sobre 12 mseg).
9. *Volcado del cache*: GALILEO utiliza la tecnología de archivos mapeados en memoria para mantener su estructura de datos. La actualización en disco de estos archivos es responsabilidad del sistema operativo y su frecuencia de actualización depende del estado de carga de la CPU y no es por tanto predecible a priori. Para intentar minimizar la pérdida de datos GALILEO fuerza un volcado a disco de dicha memoria con la frecuencia en milisegundos indicada en este parámetro.
10. *Alias*: procesador asignado en el que se ejecutará Galileo. En sistemas de 64 bits, Galileo se ejecuta dentro de un modo especial que simula los sistemas de 32 bits (Wow, "Windows on windows"). En este módo Galileo puede ser movido por Windows entre los procesadores del ordenador, provocando variaciones en el tiempo de ciclo. Para evitar esto. Galileo fija su ejecución a un único procesador (por defecto el primero). Sin embargo, dependiendo del resto de aplicaciones podría ser necesario utilizar otro diferente. Este cambio se puede realizar en caliente, es decir, sin detener Galileo.
11. *Procesador asignado*: con este parámetro se puede forzar a Galileo a utilizar una identidad distinta. Es decir, se puede forzar a que utiliza un nombre de computador distinto al que del propio PC.
12. *Flancos modo antiguo*: permite utilizar el sistema antiguo de flancos. Consultar apartado "Flancos en Galileo".
13. *URL*: Este parámetro refleja la URL de la página de actualizaciones de GALILEO. Es la página a la que se conectará el sistema de forma automática para verificar si existen versiones nuevas de GALILEO.

- *Perfilador*



1. *Perfilador*: desde esta sección se puede activar un modo de ejecución especial de Galileo. En este modo (modo perfilador), Galileo recoge los tiempos de ejecución de las maquinas, etapas y transiciones. De

esta manera se pueden detectar problemas ocasionados por etapas que consuman demasiado tiempo. Las opciones de esta sección son:

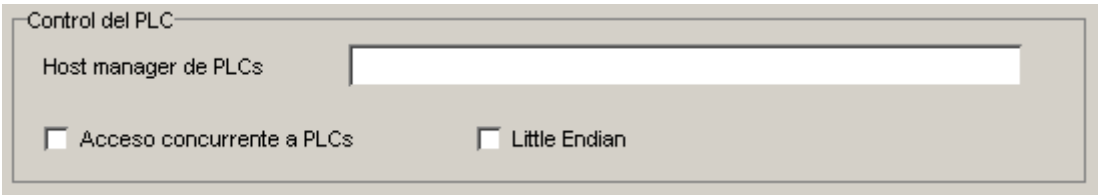
→ Utilizar Perfilador**:** activando este check, Galileo activará el perfilador. **Nunca debería esta activado en una ejecución normal, ya que se ralentizará el ciclo de ejecución**. Activando esta opción se mostrarán el resto disponibles:

I. Máquinas**:** con esta opción, el perfilador mostrará el tiempo de ejecución de cada máquina, así como los tiempos totales. Este tiempo se refiere al tiempo que Galileo emplea en ejecutar todo el código asociado a esa maquina en ese ciclo. Es posible filtrar máquinas con tiempos de ejecución bajo. Para ello se utiliza el valor (en ms) configurado en **Tiempo de aviso**. De esta manera se evita que los mensajes generados sean demasiados, y se desprecian las maquinas con tiempos bajos, que posiblemente no estén afectando en exceso al ciclo de Galileo. Es posible también visualizar la información de una única máquina. Para ello, basta con escribir el nombre de la máquina en el campo **Maquina.** Esta opción es muy útil si combina con la opción **funciones** que se explica a continuación, de manera que podamos estudiar únicamente los métodos asociados a una determinada máquina.

II. Funciones**:** con esta opción, el perfilador mostrará el tiempo de ejecución de cada función invocada. Al igual que en el caso de la opción **Maquinas**, se pueden filtrar funciones con tiempos bajos mediante el campo **Tiempo de aviso**. Está opción es la que mas sobrecarga añade a la ejecución, por lo que es conveniente utilizarla en combinación con la opción **Máquinas**, filtrando por una única máquina. Si no se hiciese así, se mostraría información de todas las maquinas en ejecución y de todas sus funciones, por lo que la información seria poco manejable.

III. Forzar**:** permite cambiar la configuración del perfilador en tiempo de ejecución sin necesidad de reiniciar Galileo. Hay que tener en cuenta que cuando se pulsa este botón, **se guarda la configuración actual del perfilador**.

11. Control del PLC



a. *Host Manager PLCs*: A partir de la versión 3.1 de Galileo, que incluye la posibilidad de comunicarse con PLCs, aparece este campo que sirve para configurar cual es el computador que actúa como Manager de PLCs para la configuración de MECALUXNet (que debe ser instalado previamente) que nos permitirá tener acceso a la red de PLCs.

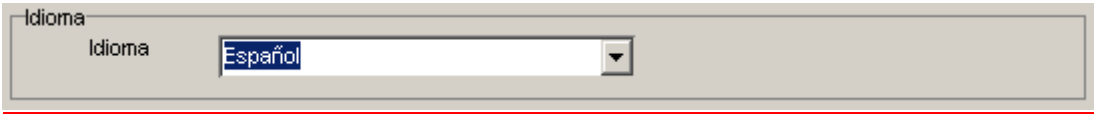
b. *Acceso concurrente a PLCs*: este parámetro permite que las lecturas de datos desde varios PLCs se realicen de manera concurrente, con un incremento de la velocidad de lectura. **Solo se puede activar si se tiene instalado la versión 10.0.1.0 o superior de MecaluxNet, ya que las versiones anteriores no soportan este tipo de comunicaciones.**

c. *Little Endian:* Indica si las comunicaciones con el PLC a través del Manager de PLCs serán mapeadas como little endian.

   Para el caso de un PLC de *Siemens*, como su mapeado es big endian, el check ha de permanecer desactivado (valor por defecto).

   Para, por ejemplo, el caso de un PLC de *Allen Bradley* ha de ser marchado el check ya que su mapeado es little endian.

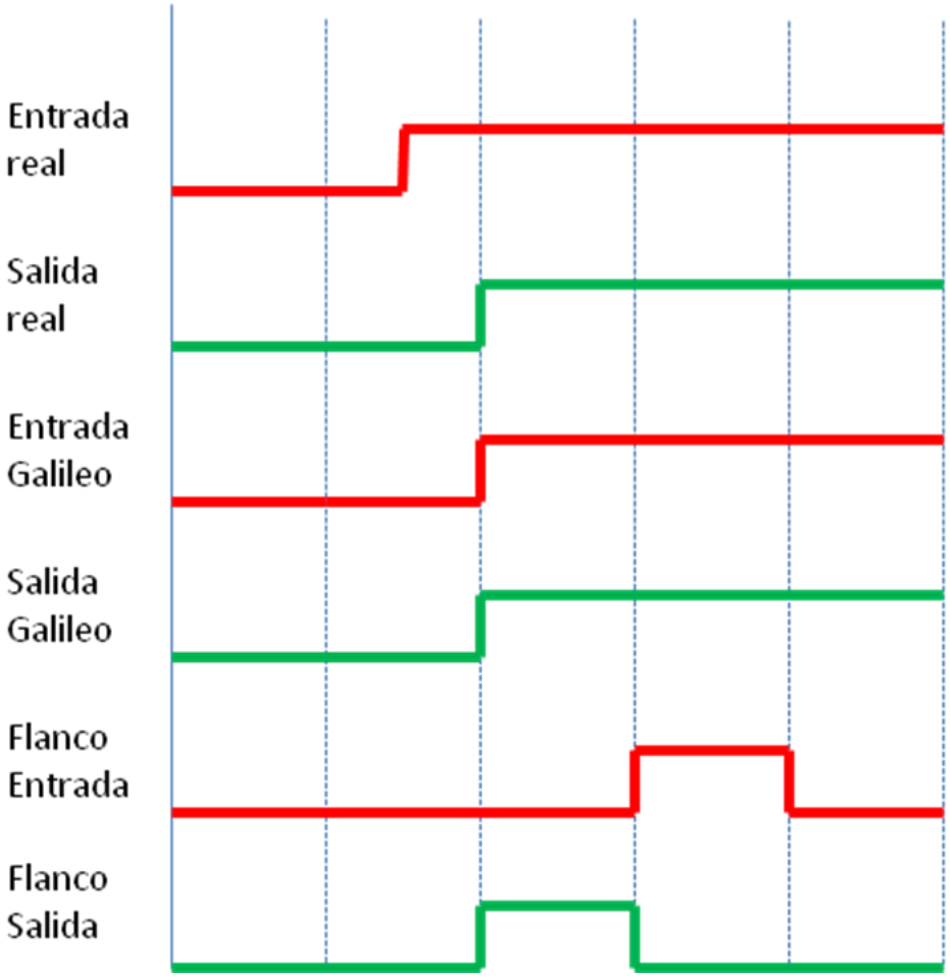
12. Idioma



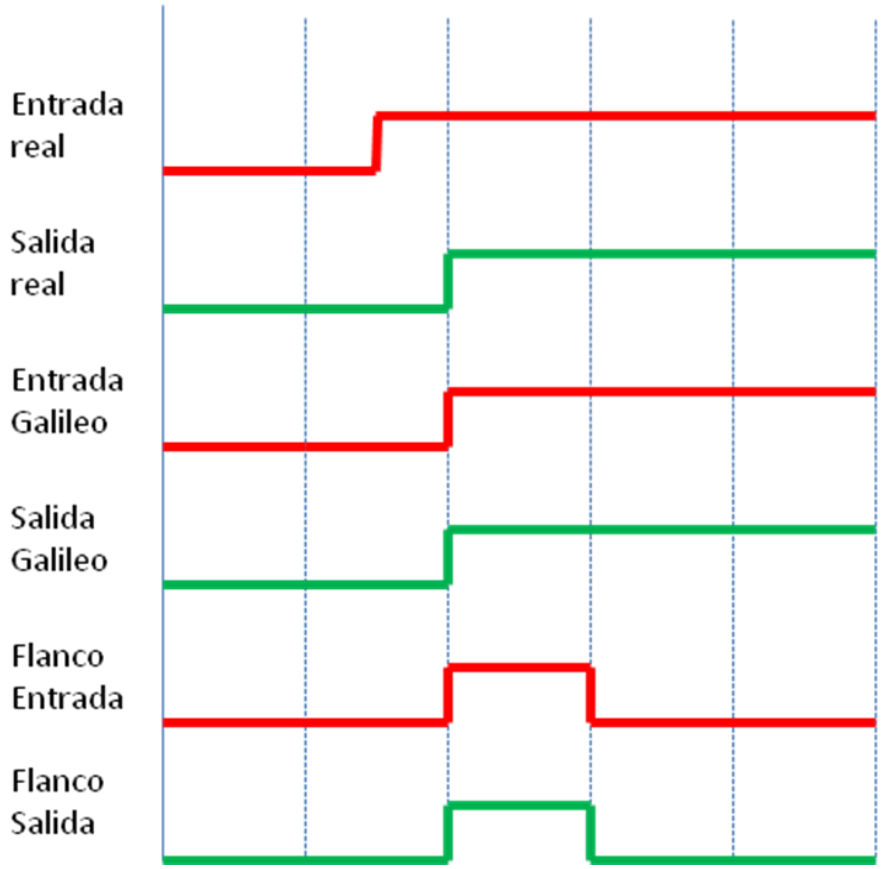
a. *Idioma:* Permite configurar el idioma de la consola de Galileo.

2.2.2.5.1 Flancos en Galileo

Hasta la versión 3.1.4.65 de Galileo, el cálculo de flancos se realizaba antes de leer las entradas desde el bus. Esto implicaba que un flanco en una entrada es detectado con un ciclo de retraso. Sin embargo, un flanco en una salida es detectado en el mismo ciclo. En el siguiente diagrama se ve esta situación:



Con el nuevo sistema, el cálculo del flanco se calcula tras la lectura de entradas, por lo que la nueva situación es la siguiente:



2.2.2.5.2 Utilización del perfilador de Galileo

Un perfilador es una aplicación, que entre otras cosas, permite conocer datos acerca del tiempo de ejecución de un programa, y poder así detectar posibles cuellos de botella o funciones ineficientes. Galileo incorpora un perfilador interno que, cuando se activa, instrumentaliza la ejecución del programa de control, calculando los tiempos de ejecución de:

- *Maquinas*: tiempo que tarda en ejecutarse una maquina en un ciclo, así como el conjunto de todas las máquinas.
- *Etapas*: tiempo que tarda en ejecutarse cada una de las funciones asociadas a una etapa.
- *Transiciones*: tiempo que tarda en ejecutarse cada una de las funciones asociadas a una transición.

En una ejecución normal de Galileo, **NUNCA SE DEBERÍA ACTIVAR EL PERFILADOR**. Esta opción ralentiza la ejecución del ciclo de ejecución de Galileo, ya que se añade código para calcular los tiempos y mostrar los resultados. Sin embargo, ante situaciones especiales, durante la depuración del programa, es posible que se necesite conocer estos datos. El caso más típico se da cuando, por ejemplo, se detectan tiempos de ciclo alto. Esto puede ser debido a varias causas, entre ellas:

- Problemas en el intercambio de E/S

- Saturación del computador
- Funciones del programa de control ineficientes

El perfilador permite detectar fácilmente este último caso, permitiendo al programador centrarse en esta función hasta que quede optimizada.

El proceso para perfilar correctamente un programa sería el siguiente:

1. Configurar, inicialmente, el perfilador para que solo sondee maquinas.
2. Una vez tengamos identificadas las máquinas que mas tiempo consumen, perfilaremos cada una de manera independiente, marcando la opción de perfilar funciones.
3. En el log *Profiler.log*, se mostrarán los datos de tiempos, maquinas y funciones que los producen y tiempo total de ciclo. Con estos datos, se revisará cada función hasta conseguir unos tiempos aceptables.

Puesto que el perfilador hace aumentar el tiempo de ciclo y escribe una cantidad nada despreciable de logs, se ha añadido como seguridad un control de tiempo de uso. Esta herramienta está pensada para realizar pruebas y no para que un proyecto se ejecute durante mucho tiempo con el perfilador activo, por lo que **si se mantiene el perfilador activo (o sin cambios) durante mas de 15 min., Galileo detendrá su ejecución y mostrará un mensaje informando de este suceso.**

2.2.2.5.2.1 Interpretar los resultados del perfilador

Los resultados del perfilador se escriben en el archivo Profiler.log, en la carpeta LOGS de Galileo. El archivo está formateado mediante campos separados por tabuladores, de manera que en caso de ser necesario pueda ser editado y analizado con una hoja de cálculo, como, por ejemplo Excel. A continuación se muestra el proceso completo de perfilado de una aplicación que tiene un tiempo de ciclo alto.

2.2.2.5.2.1.1 Perfilado inicial por máquinas

Inicialmente identificaremos la máquina o máquinas que más tiempo aportan al total del tiempo de ciclo. Para ello configuramos el perfilador únicamente con las opciones "*Utilizar perfilador*" y "*Máquinas*". Con figuramos un tiempo de aviso de 1 ms, y no pondremos nada en el campo **Maquina** para que las muestre todas. Con esta configuración, en el archivo Profiler.log obtenemos los siguiente (solo se muestra un parte):

```
15:02:08.300 29 Jan 2009- MACHINE(TM_8)      1        ms
15:02:08.300 29 Jan 2009- MACHINE(TCF_64)    1        ms
15:02:08.300 29 Jan 2009- MACHINE(TRF_274)   1        ms
15:02:08.300 29 Jan 2009- MACHINE(TRT_102)   1        ms
15:02:08.300 29 Jan 2009- MACHINE(PUPITRE_21)        1       ms
15:02:08.316 29 Jan 2009- MACHINE(PUPITRE_22)        1       ms
15:02:08.316 29 Jan 2009- MACHINE(PUPITRE_23)        1       ms
15:02:08.316 29 Jan 2009- MACHINE(PUPITRE_24)        1       ms
15:02:08.316 29 Jan 2009- MACHINE(PUPITRE_25)        1       ms
15:02:08.316 29 Jan 2009- MACHINE(TCT_211)   6        ms
15:02:08.316 29 Jan 2009- MACHINE(TCT_212)   6        ms
```

```
15:02:08.332 29 Jan 2009- MACHINE(TCT_213) 7        ms
15:02:08.332 29 Jan 2009- MACHINE(TCT_214) 5        ms
15:02:08.347 29 Jan 2009- MACHINE(TCT_215) 7        ms
15:02:08.347 29 Jan 2009- MACHINE(TCT_216) 6        ms
15:02:08.363 29 Jan 2009- MACHINE(TCT_217) 7        ms
15:02:08.363 29 Jan 2009- MACHINE(TCT_218) 6        ms
15:02:08.363 29 Jan 2009- MACHINE(TCF_197) 1        ms
15:02:08.363 29 Jan 2009- Machines execution TOTAL TIME:      60      ms
```

En este log, se ve que el tiempo total gastado en la ejecución del código es de 60 mseg. Comparando este valor con el del tiempo de ciclo (alrededor de 65 ms), vemos que el problema está en el propio código, ya que la mayor parte del tiempo se gasta en él. Además se ve que las maquinas TCT_xxx, son las que mas tiempo consumen, siendo las que tendríamos que perfilar.

Revisando el proyecto, todas las máquinas TCT utilizan el mismo secuenciador, asi que perfilaremos una cualquiera.

2.2.2.5.2.1.2 Perfilado de una máquinas por funciones

Configuraremos el perfilador, para que estudie, únicamente, la maquina TCT_213. Para ello, marcamos las opciones "*Utilizar perfilador*", "*Maquinas*" y "*Funciones*". Asignamos un *tiempo de aviso* a máquinas y funciones de 1 ms, y en el campo ***Máquina*** escribimos TCT213. El resultado, en el archivo Profiler.log es el siguiente:

```
15:10:27.930 29 Jan 2009- TCT_213.ET_Anterior_Transportador     6       ms
15:10:27.930 29 Jan 2009- MACHINE(TCT_213) 7        ms
15:10:27.977 29 Jan 2009- Machines execution TOTAL TIME:      7       ms
```

En este caso, el secuenciador únicamente esta ejecutando la función *ET_Anterior_Transportador*, por lo que esta es la única que se muestra, con un consumo de 6 ms. El tiempo de ejecución completa de la máquina es de 7 mseg., y el último dato mostrado, en este caso representa el tiempo total gastado en ejecutar todas las máquinas perfiladas, que como en este caso solo es una, coincide con el tiempo de la propia máquina. Este ejemplo es muy sencillo, pero se ve como el problema está localizado en la función *ET_Anterior_Transportador*, que consume demasiado. En este momento habría que estudiar que hace esta función y como se puede modificar para que consuma menos.

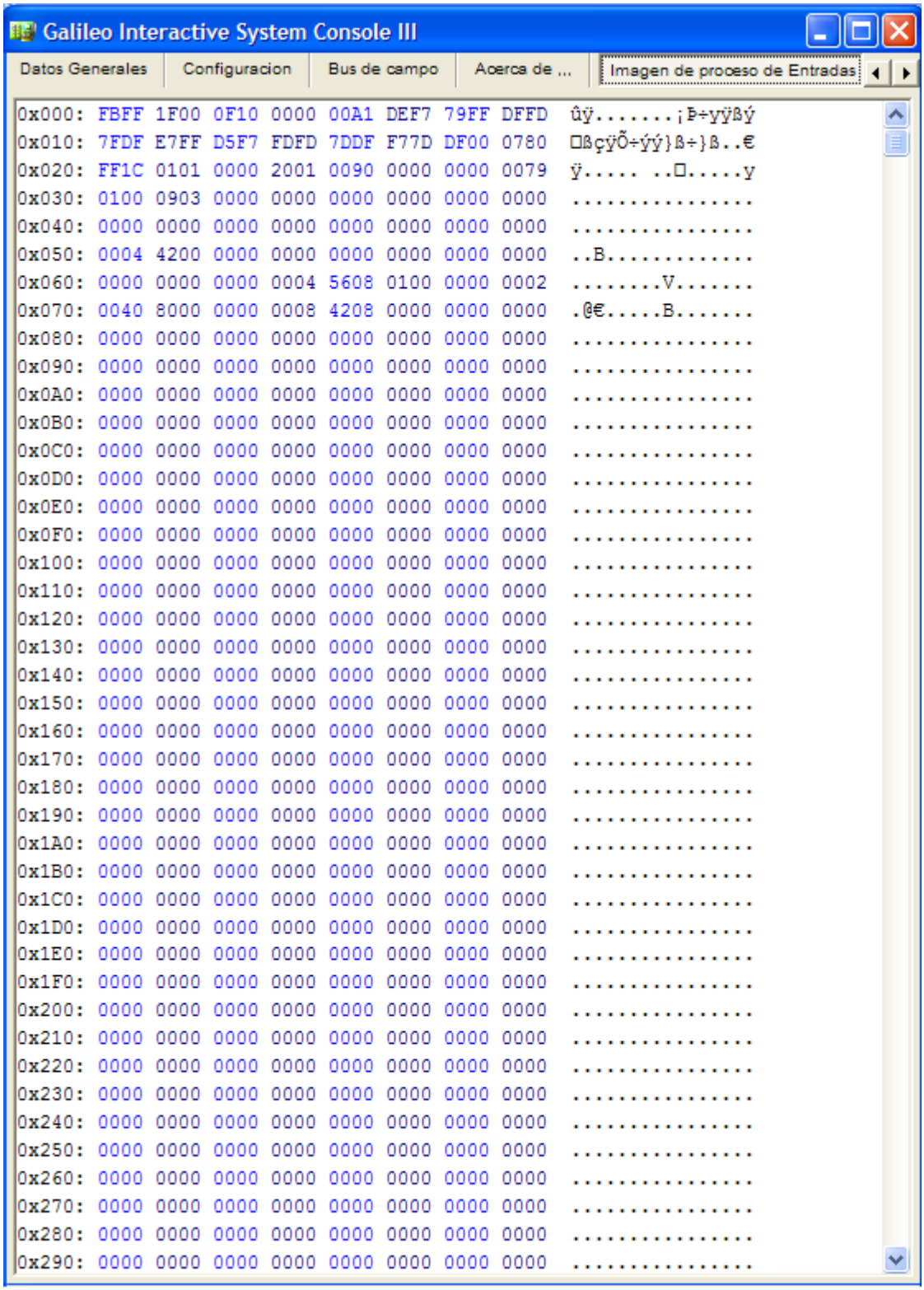
En este ejemplo, el problema estaba en que se llamaba desde dentro de la función a funciones de manera innecesaria. Una vez modificada, los resultados del perfilador son los siguientes:

```
13:17:55.201 29 Jan 2009- MACHINE(TCF_14)    1       ms
13:17:55.201 29 Jan 2009- MACHINE(TM_106)    1       ms
13:17:55.201 29 Jan 2009- MACHINE(TM_113)    1       ms
13:17:55.201 29 Jan 2009- MACHINE(TM_264)    1       ms
13:17:55.217 29 Jan 2009- MACHINE(TCF_229)   1       ms
13:17:55.217 29 Jan 2009- MACHINE(PUPITRE_21)        1       ms
13:17:55.217 29 Jan 2009- MACHINE(PUPITRE_22)        1       ms
13:17:55.217 29 Jan 2009- MACHINE(PUPITRE_23)        1       ms
13:17:55.217 29 Jan 2009- MACHINE(PUPITRE_24)        1       ms
13:17:55.217 29 Jan 2009- MACHINE(TCT_211)   1       ms
13:17:55.217 29 Jan 2009- MACHINE(TCT_213)   1       ms
13:17:55.217 29 Jan 2009- MACHINE(TCT_214)   1       ms
13:17:55.217 29 Jan 2009- MACHINE(TCT_216)   1       ms
13:17:55.217 29 Jan 2009- MACHINE(TCT_218)   1       ms
13:17:55.217 29 Jan 2009- Machines execution TOTAL TIME:    14      ms
```

Como se ve, se ha mejorado considerablemente los tiempos de ejecución.

2.2.2.6 Visualizar las imágenes de proceso

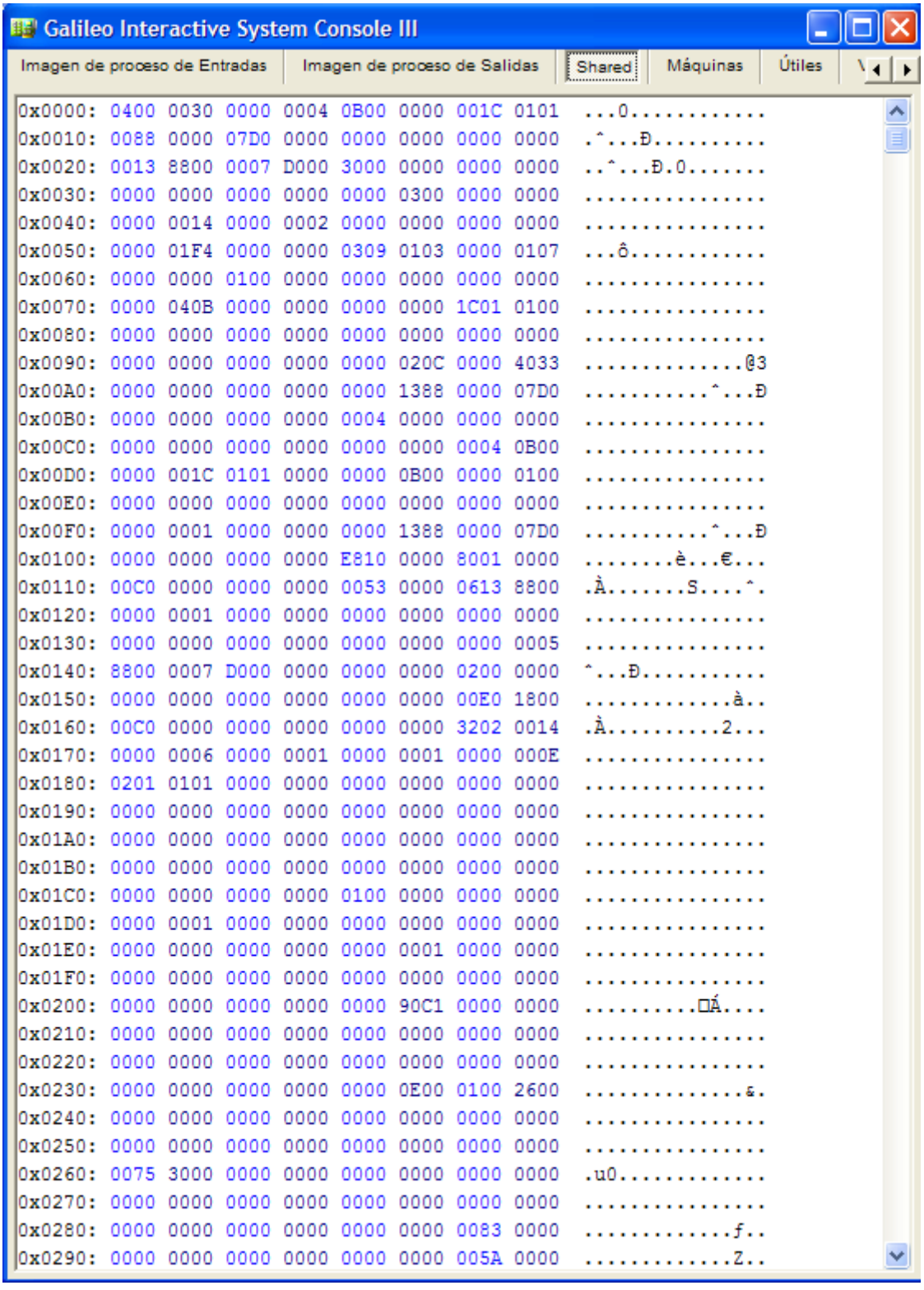
Existen también un par de solapas que nos permiten ver directamente los valores (en hexadecimal) que se encuentran en la memoria de proceso de entradas y salidas. Estos mapas son solo de lectura, y son un volcado directo de lo que se esta viendo desde la tarjeta.



La información así mostrada corresponde con la ordenación posterior a la reubicación de `bytes` entre tarjetas que se configura con el entorno de desarrollo en el Designer.

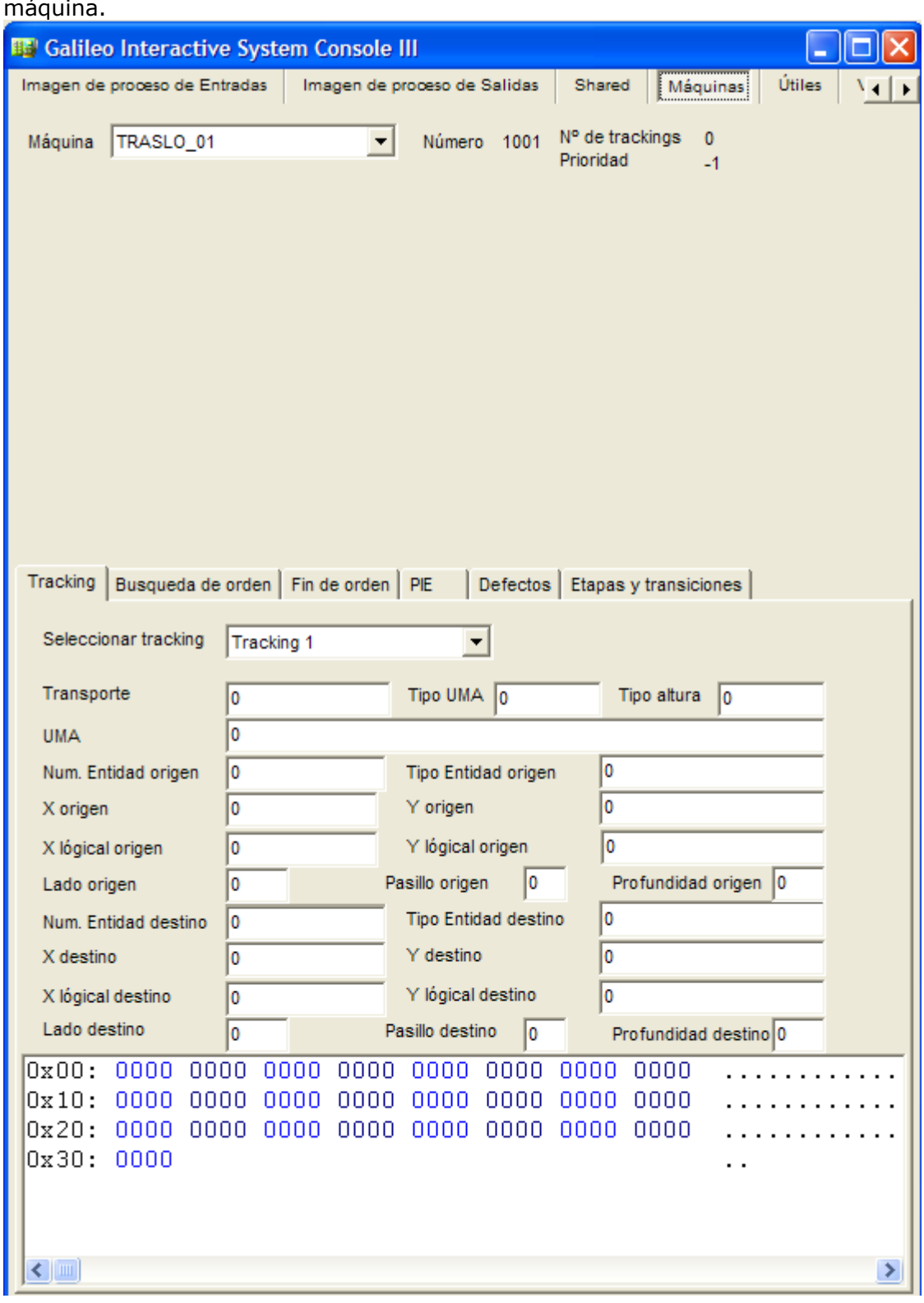
2.2.2.7 Visualizar memoria shared

Existe una solapa que nos permite ver los datos escritos en la memoria shared, es decir, en la variables shared de lectura o lectura/escritura que se definan en el programa. Esta solapa es de solo lectura, y es un volcado de la memoria shared que está utilizando Galileo.



2.2.2.8 Datos de las máquinas

Desde esta solapa también es posible visualizar el comportamiento de una máquina.



En la parte superior izquierda, un campo nos permite seleccionar la máquina a visualizar.

Más a la derecha, unas etiquetas nos informan de número de trackings activos de la máquina, así como su prioridad de ejecución actual. Siguiendo hacia abajo, nos encontramos un control de páginas que nos da diferentes vistas de los datos de la máquina.

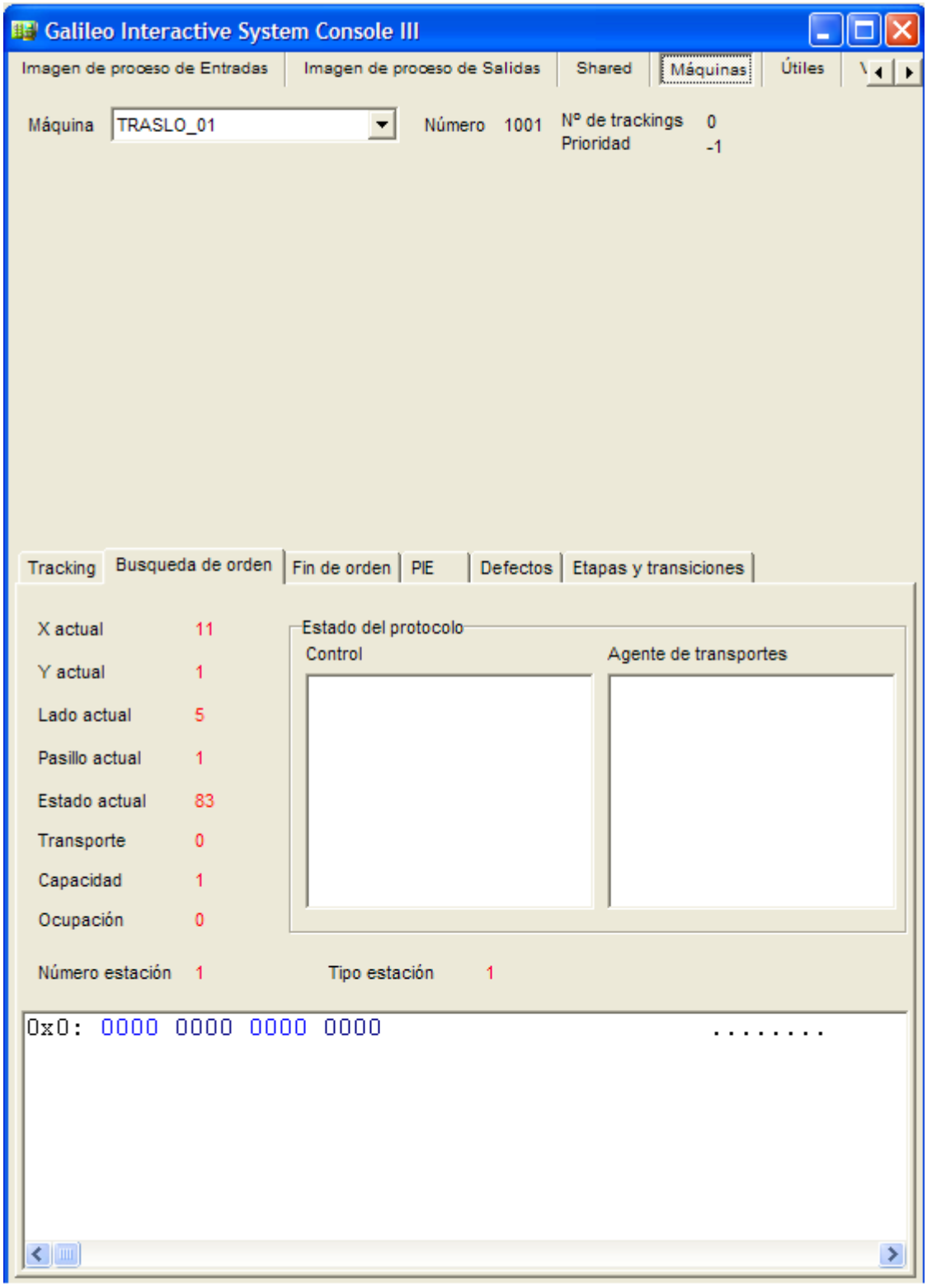
2.2.2.8.1 Consultar trackings

Esta solapa nos muestra la información relativa a los datos del tracking seleccionado. Dicha información es numérica (no se muestran las entidades lógicas asociadas a los valores numéricos). Se muestran también mediante un control hexadecimal los valores de los CUSTOM_DATA de dicha máquina.

2.2.2.8.2 Búsqueda de orden

Esta solapa permite mostrar los datos referidos a las peticiones de orden que pudiera realizar la máquina. Nótese que estos datos sólo tendrán sentido si la máquina realiza dichas peticiones en el código.

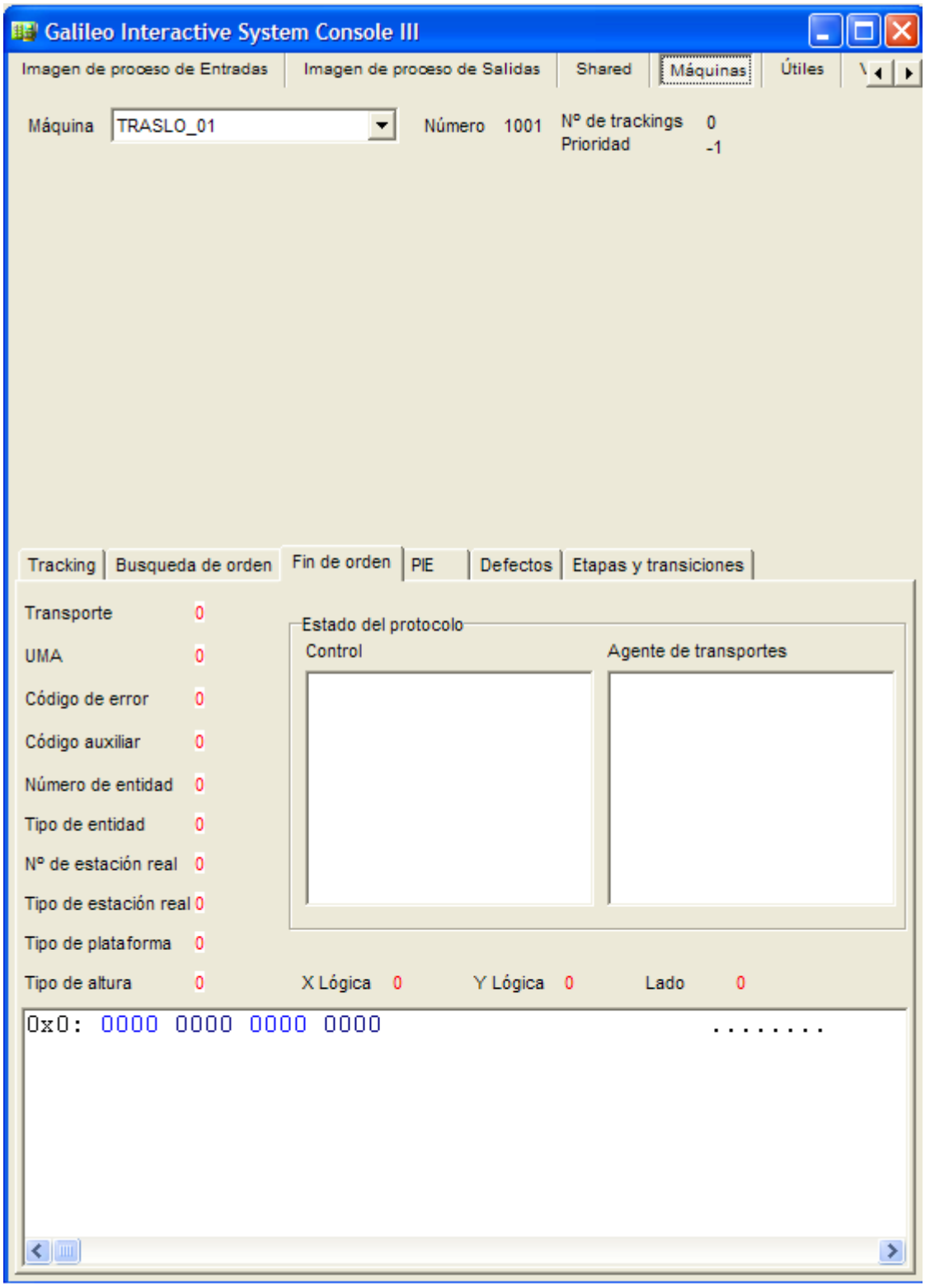
Como novedad sobre la versión II los flags de protocolo entre control y agente de transportes se muestran de manera que son comprensibles por el usuario, al describirse de manera textual en vez de numérica.



2.2.2.8.3 Zona de Fin de Orden

Aquí se muestran los datos referidos a los fines de orden que han llegado a la máquina. Nótese que estos datos sólo tendrán sentido si en dicha máquina se

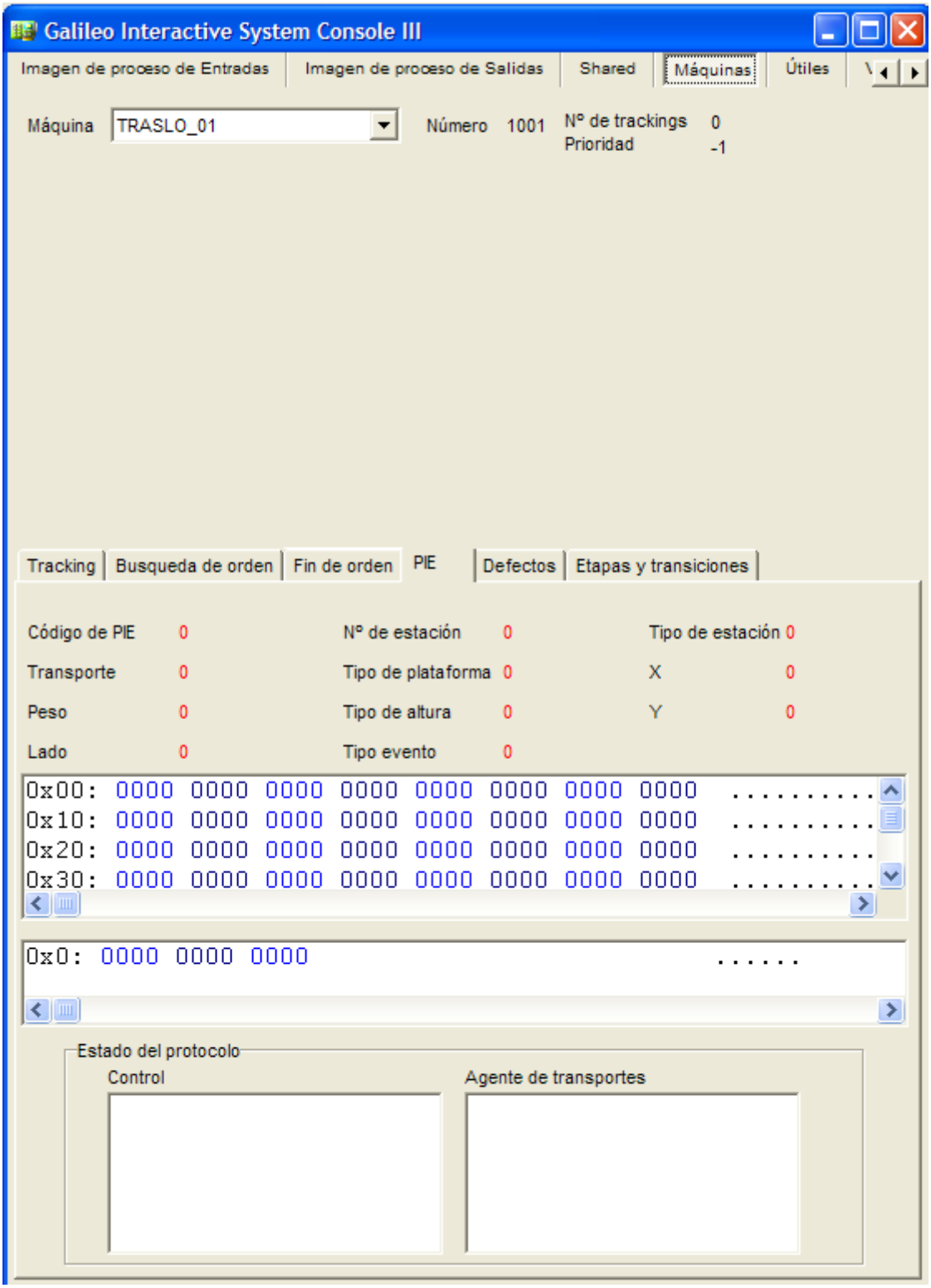
finalizan ordenes. De la misma manera que en la búsqueda de orden los flag de comunicaciones se muestran en forma de texto comprensible por el usuario.



2.2.2.8.4 Zona de PIE

Aquí se muestran los datos de PIE que ha realizado la máquina. Nótese que estos datos sólo tendrán sentido si la máquina efectivamente es un Punto de Identificación de Entrada (PIE).

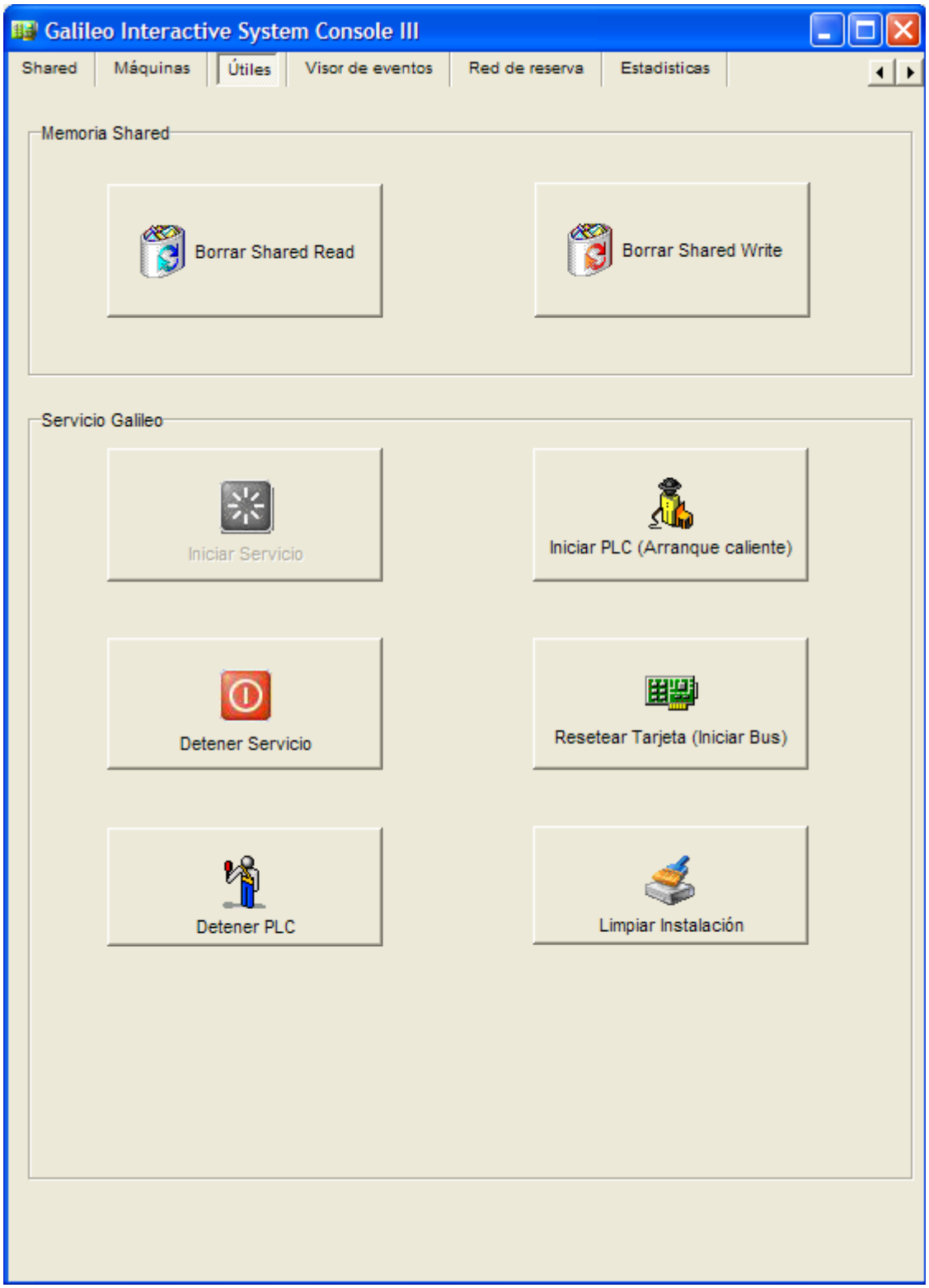
Además de estos grupos, en la parte superior de estas pestañas pueden aparecer un par de etiquetas en rojo mostrando los mensajes de estado y error por los que va pasando la máquina.



Toda esta información se refresca de forma automática y periódica.

2.2.2.9 Solapa Útiles

Bajo esta solapa etiquetada como "Útiles" nos encontramos lo siguiente:

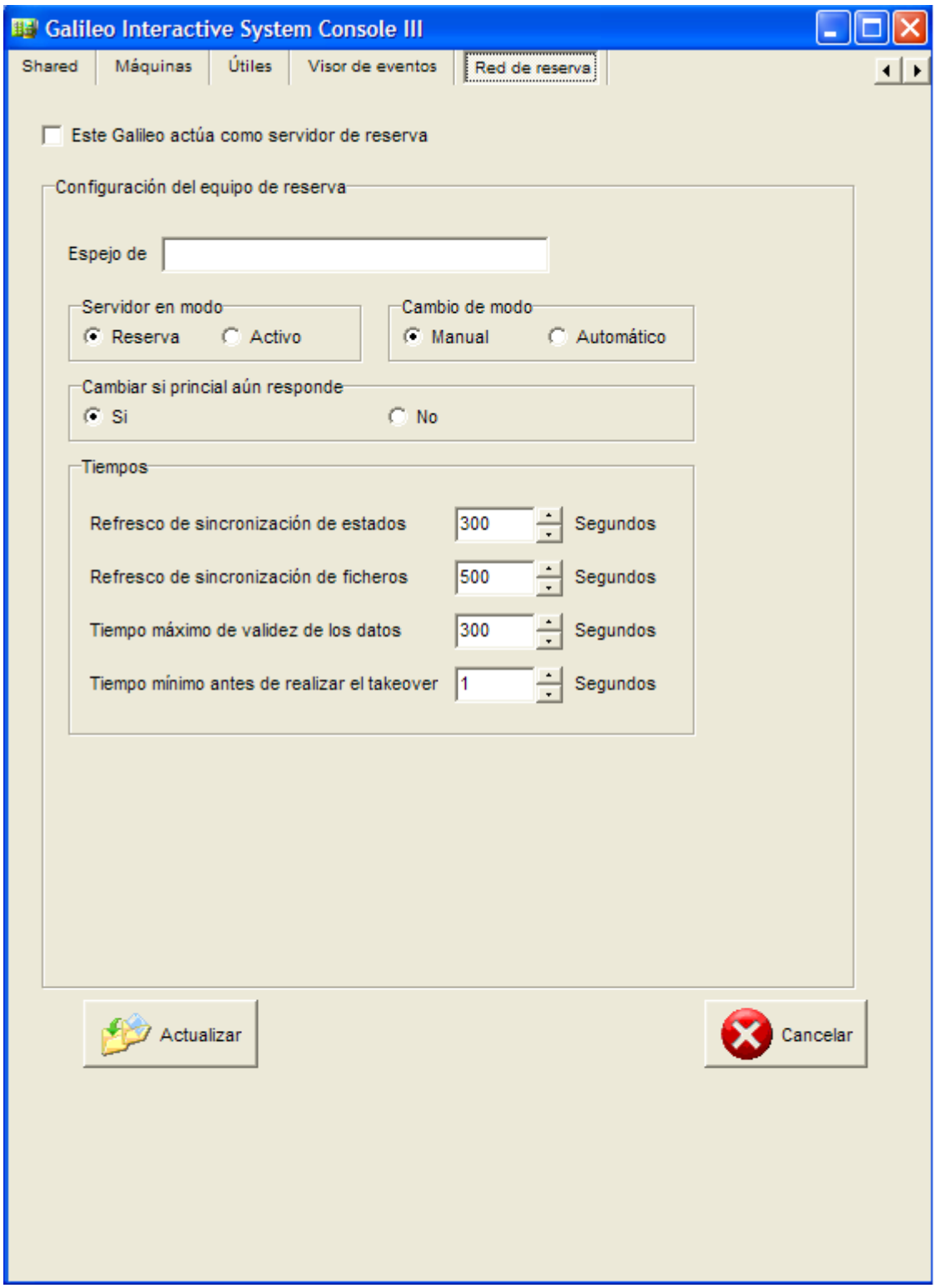


A continuación explicaremos para que se utilizan cada uno de estos botones:

- Borrar Shared Read → Borra la memoria compartida de lectura que usan todas las variables que no son de periferia.
- Borra Shared Write → Borra la memoria compartida de escritura que usan todas las variables que no son de periferia.
- Iniciar Servicio → Permite Iniciar el servicio Galileo desde aquí, causando un arranque en frío.
- Detener Servicio → Provoca la parada completa del Servicio Galileo.
- Iniciar PLC → Provoca un Arranque caliente del programa de control.
- Resetear Tarjeta (Iniciar Bus) → Al igual que el anterior, provoca un reinicio del bus y fuerza un arranque en caliente.
- Detener PLC → Detiene el programa de control. El servicio sigue ejecutándose, pero el programa de control se detiene y no se realizan más intercambios de entrada / salida hasta que se haga un arranque en caliente o se reinicie el servicio por completo.
- Limpiar Instalación → Este botón permite limpiar de forma completa la memoria de control. Por consiguiente provoca el reseteo de todas las máquinas a sus etapas iniciales.

2.2.2.10 Solapa "Red de Reserva"

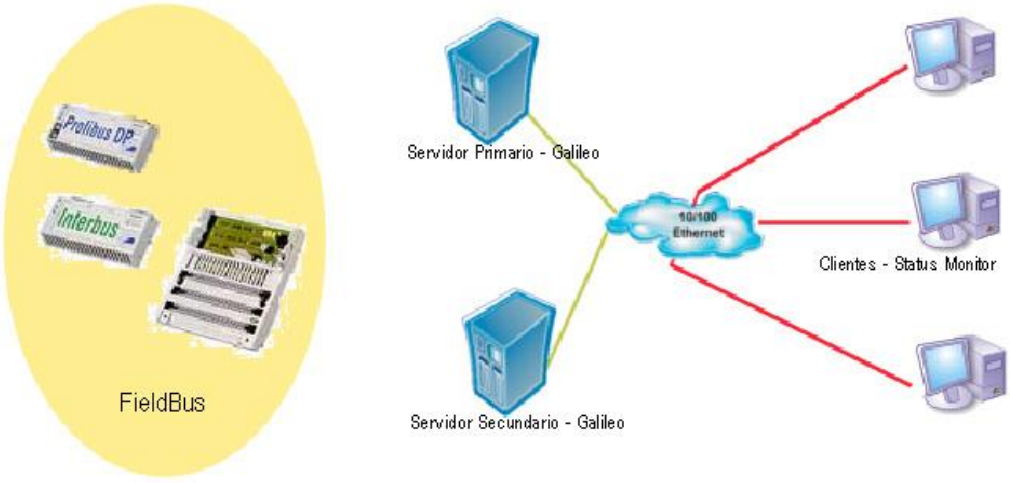
Existe una solapa con esta etiqueta donde podemos observar la siguiente configuración:



Esta pantalla tiene sentido si deseamos que el servicio Galileo del computador en el que estamos actúe como servidor de reserva o respaldo de otro equipo de la red Galileo, de forma que ante una eventual caída del principal, este tome el control del bus de campo e intente recuperar el funcionamiento habitual de la instalación.

Para activar esta opción, simplemente basta con activar la marca de "*Este Galileo Actúa como Servidor de reserva*", y rellenar el campo "*Espejo de*" con el nombre del computador que queramos que sea monitorizado. Al iniciar esta configuración, el modo del servidor debe establecerse también como "Reserva", para que al menos obtenga los datos del programa y ficheros de configuración del equipo principal al arrancarlo la primera vez. Si el campo "*Servidor en Modo*" se establece a "*Activo*", y posteriormente se reinicia el servicio Galileo, este intentará tomar el control del Bus de Campo y pasar a funcionar como si fuera el equipo Primario. En modo "*Reserva*", el servicio Galileo intentará sincronizar sus datos con los del servidor primario, a fin de tener una imagen actualizada de la instalación a usar en caso de sustitución del sistema primario.

En la siguiente imagen podemos observar un esquema de una configuración posible.



Los computadores con el servicio Galileo están conectados al bus de campo y a su vez a una red Ethernet que les permitirá comunicarse entre si y con otros equipos que quieran ejecutar aplicaciones cliente (como el Status Monitor o el Designer) para comunicarse con ellos u obtener información del estado del bus de campo.

El bus de campo puede ser actualmente InterBus o Profibus, pero sólo con Profibus el sistema de replica de Galileo puede funcionar en modo "*Automático*".

En ambos servidores Galileo, tanto la configuración de bus como de las tarjetas de control debe ser IDÉNTICA. En el caso de Profibus, la tarjeta de control puede estar conectada también al bus de campo, con lo que podremos usar el modo de cambio automático. En el caso de Interbus, al no poder darse este caso, sólo el primario puede estar conectado al bus, y si se desea cambiar de servidor, hay que desconectar el conector de la tarjeta de control al bus y conectarlo en la tarjeta de control del servidor de reserva, realizando entonces el cambio de forma manual en esta pantalla y reiniciando el servicio.

El resto de los campos de esta pantalla tienen significado para el caso de que se active la opción de cambio automático. En este modo, el servidor de reserva está monitorizando de forma continua tanto el estado del servidor primario como el estado del bus de campo (de ahí la necesidad de que este conectado a la red

Profibus). Si seleccionamos "*Si*" en la opción "*Cambiar si el principal aun responde*" esto significa que el servidor secundario intentara tomar el control del bus en el caso de que se detecte un fallo en la tarjeta de control del primario, incluso si el servicio de control Galileo del servidor primario esta activo y respondiendo a los mensajes. En cambio, si seleccionamos "*No*", el servidor secundario solo intentara tomar el control en el caso de que el servicio Galileo del servidor Primario deje de comunicarse completamente.

La ventaja de usar el primer método es que la respuesta ante un fallo de bus será más rápida, y además añade una funcionalidad extra a la red: el servidor primario cambia su configuración "en caliente" para pasar a ser el servidor de reserva del nuevo principal, y realiza una parada controlada. En su siguiente arranque, funcionará como servidor de reserva, hasta que se produzca un nuevo cambio o hasta que se reconfigure manualmente, es decir: intercambian roles.

La desventaja de este modo también es clara: si el fallo de comunicación con el bus no es producido por un fallo en la tarjeta de control, sino por un fallo dentro de la propia estructura del bus, el cambio de servidor no solucionará el problema, y habremos realizado un cambio innecesario que será necesario revertir para volver a tener la situación original una vez se solucionen los problemas del bus. Es por ello que la opción de NO cambiar si el servidor primario aun responde es más aconsejable en la mayoría de los casos.

Los últimos 4 campos se refieren a tiempos de refresco y sincronización de datos entre el servidor primario y el de reserva, a saber:

- Refresco de sincronización de estados: Frecuencia, en segundos, a la que el servidor secundario realiza "polling" del servidor primario a fin de actualizar los estados de la instalación y la imagen de proceso del bus.
- Refresco de sincronización de ficheros: El Sistema de Control Galileo del servidor primario puede registrar modificaciones en los ficheros de control que usa para gestionar la instalación (cambios en los mapas de cotas, modificaciones al programa de control). Para evitar que dichos cambios tengan que ser exportados al servidor de reserva también, este realiza, de forma periódica, una copia de dichos ficheros desde el primario. La frecuencia de dicha copia se establece mediante esta opción. Usualmente es un valor elevado (salvo en periodos de pruebas o de cambios intensivos) una copia cada hora o cada día incluso, debería ser más que suficiente en la mayoría de los casos.
- Tiempo máximo de validez de datos: Este valor tiene significado en el caso de que se pretenda realizar un cambio automático. Si el cambio de servidores debe producirse, pero el tiempo entre la última vez que se sincronizaron los datos con el servidor primario y el de reserva supera este límite, el cambio automático no se realizará, al entender el servidor de reserva que la imagen que tiene del sistema puede ser obsoleta y por lo tanto un cambio automático podría dar lugar a problemas. Si se reinicia el Servicio o el equipo, este tiempo ya no se tiene en cuenta, lo que equivaldría a un cambio manual, por lo tanto este campo solo tiene utilidad en el caso de querer realizar cambio automático, y el valor a usar depende enteramente del tipo de instalación que se pretende gobernar.

- Tiempo mínimo antes de realizar el "takeover": Una vez que el servidor de reserva detecta las condiciones que dan pie a pensar que es necesario un cambio automático, esperará durante este periodo de tiempo a ver si el servidor primario vuelve a responder o si el problema de bus desaparece. Si pasado este tiempo las condiciones del cambio aun persisten, se procederá a realizar el cambio si el modo es "Automático". Lógicamente, este tiempo nunca debe superar al tiempo máximo de validez de datos.

Es importante notar que en modo automático, el cambio entre el servidor principal y de reserva se hace de forma relativamente rápida y silenciosa, con lo que la única manera de detectar que se ha producido dicho cambio es abrir la consola y comprobar el estado de los servicios, puesto que los clientes como el Status Monitor, el Designer, u otros Galileos en la red tampoco detectarán el cambio más allá de un breve periodo de tiempo en el que dicho servidor dejará de responder. Esto facilita sobremanera la vida del usuario de la instalación, pero puede entorpecer al programador, que no tendrá conciencia clara de donde se está ejecutando el servicio Galileo realmente.

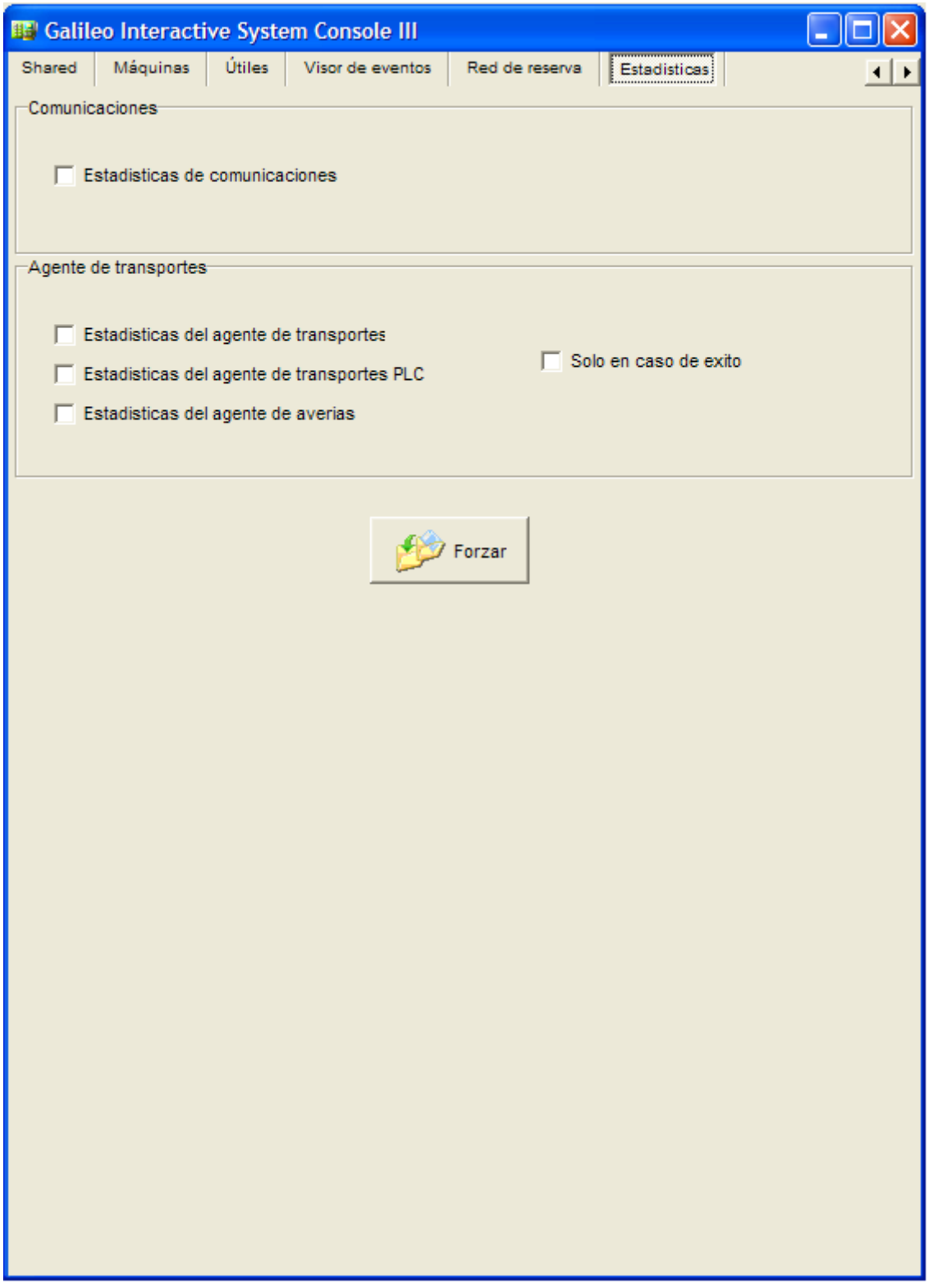
NOTA IMPORTANTE:

Es por ello que ***no se recomienda usar el modo automático mientras se están desarrollando cambios en el sistema de control, o durante cooperaciones de mantenimiento que detengan el funcionamiento normal del bus***. ***La manera de desactivar un equipo de reserva es configurando su pantalla en la consola Galileo y posteriormente reiniciar el servicio***.

Es fundamental entender que la consola, al contrario que el Designer, no realiza ninguna comunicación (prácticamente) con GALILEO. Por este motivo es capaz de visualizar todas las informaciones indicadas incluso con el servicio de GALILEO parado. En este caso estaríamos visualizando una imagen congelada en el tiempo en el cual GALILEO fue detenido. Esta imagen se puede considerar post-mortem, es decir en caso de parada ya sea natural o fallo, debiéramos visualizar los últimos datos. Especialmente útil puede resultar la visualización de las etapas y trackings de las máquinas.

2.2.2.11 Solapa "Estadísticas"

Por último aparece una solapa denominada Estadísticas, que permite obtener más información referente tanto al estado de las comunicaciones entre PCs, como a la ejecución del Agente de Transportes. Todas las estadísticas que se habilitan desde esta pestaña aparecerán dentro del log Statistic.log que se encontrará junto con el resto de logs en …\Galileo\LOGS.



Para activarlas es necesario marcar el check correspondiente y accionar el pulsador de "Forzar". De esta forma quedarán habilitadas ininterrumpidamente durante *10 minutos*, período después del cual se deshabilitarán automáticamente.

Aunque dispongamos de alguna estadística habilitada siempre que se abra la consola de Galileo, aparecerán todas deshabilitadas, por lo que no se produce refresco sobre su estado. Para poder deshabilitarlas, es necesario deseleccionarla/s y volver a pulsar el botón de "Forzar".

En el caso de que se detenga el servicio de Galileo con alguna de las estadísticas habilitada, se detendrá su ejecución. Si se desea continuar con ellas tras un nuevo arranque, es necesario volver a habilitarlas como se ha descrito.

- Comunicaciones

  - ✓ Estadísticas de comunicaciones → Se indicarán tiempos, contadores, etc. que faciliten la localización de problemas en las comunicaciones entre los distintos PCs. A continuación se detalla mediante un ejemplo la parte del fichero Statistic.log que hace referencia a las comunicaciones:

```
12.43.48.300:19 Jan 2010:
GalileoMessenger::Statistics (in ms) → Entrada de la parte de comunicaciones
==================================
Remote computers(1) → Número de computadores remotos con los que existe
                      comunicación
SleepTime(100.000000) → Tiempo entre ciclos de refresco de periferia.
                        Configurado en la pestaña "Configuración" de la
                        consola de Galileo con el  parámetro "Refresco
                        periferia" (por defecto se encuentra configurado a
                        100)
Last Comms:
--------------
       GALILEO:
              LastComm(105.000000) → Tiempo de la última comunicación
              Error(NO) → Indica si existe o no error en ese ciclo de
                          comunicación
              MaxErrorTime(0.000000) → Tiempo máximo en el ciclo con error
              MaxSuccessTime(306.000000) → Tiempo máximo de comunicación
              AverageSuccessTime(0.038941) → Tiempo medio de comunicación
              ErrorCounter(0) → Contador de ciclos con error
              SuccessCounter(2567) *-> Contador de ciclos sin error*
12.43.48.394:19 Jan 2010:
GalileoMessenger::Statistics (in ms)
==================================
Remote computers(1)
SleepTime(101.000000)
Last Comms:
--------------
       GALILEO:
              LastComm(5031.000000)
              Error(YES)
              MaxErrorTime(5031.000000)
              MaxSuccessTime(306.000000)
              AverageSuccessTime(0.039315)
              ErrorCounter(1)
              SuccessCounter(2567)
```

- Agente de Transportes

  ✓ Estadísticas del agente de transportes → Activa las estadísticas referentes a tiempos, errores, contadores, etc. relativas al Agente de Transportes con un sistema de control mediante Galileo. A continuación se detalla mediante un ejemplo cada uno de los campos que aparecen:

```
12:42:54.653 19 Mar 2010-
TransportAgent31::Statistics (in ms) → Entrada de la parte del Agente de
==================================     Transportes con sistema de control
                                       Galileo
WorkTime(1001.000000)SleepTime(1000.000000) → Tiempo en ejecutar las
                                              operaciones (en ms) / Tiempo de
                                              descanso
Current Cycle:
--------------
      Number of machines(1) → Número de máquinas del computador
      TotalIISTime(0.000000) → Tiempo (en ms) en gestionar todos los eventos
            Success counter(0) → Número de eventos con éxito
            Error counter(1)   → Número de eventos con error
            MaxIISTime(0.000000) → Tiempo (en ms) en gestionar el evento más
                                   lento
            WaitTime(0) → Tiempo (en ms) de espera tras procesar los eventos
            WaitLoopCompleted(YES) → Indica si se completó el tiempo de
                                     espera máximo o no
      EndCommandTime(0.000000) → Tiempo (en ms) en gestionar todos los fines
                                 de orden
            Success counter(0) → Número de fines de orden con exito
            Error counter(1)   → Número de fines de orden con error
            MaxEndCommandTime(0.000000) → Tiempo (en ms) en gestionar el fin
                                          de orden más lento
            WaitTime(0) → Tiempo (en ms) de espera tras procesar los fines
                          de orden
            WaitLoopCompleted(YES) → Indica si se completó el tiempo de
                                     espera máximo o no
      SearchTime(0.000000) → Tiempo (en ms) en gestionar todas las búsquedas
                             de orden
            Success counter(0) → Número de búsquedas de orden con éxito
            Error counter(1) → Número de búsquedas de orden con error
            MaxSearchTime(0.000000) → Tiempo (en ms) en gestionar la
                                      búsqueda de orden mas lenta
      UpdateStationTime(0.000000)      UpdateRoutesTime(0.000000)→ Tiempo (en
ms) en gestionar las actualizaciones (estaciones/rutas)
      CancelRequestTime(1.000000)      CancelReponseTime(0.000000)   → Tiempo
(en ms) en gestionar las cancelaciones (petición/respuesta)
      Time in DB(0.000000) → Tiempo total en base de datos (suma de los
                             tiempos individuales)
Stored data:
------------
      MaxIIS(0.000000) → Tiempo (en ms) del evento más lento
      MaxEndCommand(0.000000) → Tiempo (en ms) del fin de orden más lento
      MaxSearchWithCommand(0.000000) → Tiempo (en ms) de la búsqueda de orden
                                       (con éxito) más lenta
      MaxSearchWithoutCommand(0.000000) → Tiempo (en ms) de la búsqueda de
```

```
                                                    orden (sin éxito) más lenta
      MaxUpdateStations(0.000000) → Tiempo (en ms) más lento en gestionar la
                                    actualización de estaciones
      MaxUpdateRoutes(0.000000) → Tiempo (en ms) más lento en gestionar la
                                  actualizacion de rutas
      MaxTimeInDB(0.000000) → Tiempo (en ms) más en base de datos
```

- ✓ Estadísticas del agente de transportes PLC → Activa las estadísticas referentes a tiempos, errores, contadores, etc. relativas al Agente de Transportes con un sistema de control mediante PLC comercial. A continuación se detalla mediante un ejemplo cada uno de los campos que aparecen:

```
12:42:55.215 19 Mar 2010-
PLCTransportAgent31::Statistics (in ms) → Entrada de la parte del Agente de
==================================      Transportes con sistema de control
                                        Galileo
WorkTime(1001.000000)      SleepTime(1000.000000)→Tiempo en ejecutar las
                                                  operaciones (en ms) / Tiempo
                                                  de descanso
Current Cycle:
--------------
      Number of machines(1) → Número de máquinas del computador
      TotalIISTime(0.000000)  → Tiempo (en ms) en gestionar todos los eventos
            Success counter(0) → Número de eventos con éxito
            Error counter(1) → Número de eventos con error
            MaxIISTime(0.000000) → Tiempo (en ms) en gestionar el evento más
                                   lento
      EndCommandTime(0.000000) → Tiempo (en ms) en gestionar todos los fines
                                 de orden
            Success counter(0) →Número de fines de orden con éxito
            Error counter(1) →Número de fines de orden con error
            MaxEndCommandTime(0.000000) → Tiempo (en ms) en gestionar el fin
                                          de orden más lento
      SearchTime(0.000000) → Tiempo (en ms) en gestionar todas las búsquedas
                             de orden
            Success counter(0) → Número de búsquedas de orden con éxito
            Error counter(1) → Número de búsquedas de orden con error
            MaxSearchTime(0.000000) → Tiempo (en ms) en gestionar la
                                      búsqueda de orden mas lenta
      UpdateStationTime(0.000000) → Tiempo (en ms) en gestionar la
                                    actualización de estaciones
      CancelRequestTime(1.000000)      CancelReponseTime(0.000000)→     Tiempo
(en ms) en gestionar las cancelaciones (petición/respuesta)
      Time in DB(0.000000) → Tiempo total en base de datos (suma de los
                             tiempos individuales)
  PLCReadTime(815.000000) → Tiempo (en ms) en leer la información de todos los
                            PLCs
Stored data:
------------
      MaxIIS(0.000000) → Tiempo (en ms) del evento más lento
      MaxEndCommand(0.000000) → Tiempo (en ms) del fin de orden más lento
      MaxSearchWithCommand(0.000000) → Tiempo (en ms) de la búsqueda de orden
                                       (con éxito) más lenta
      MaxSearchWithoutCommand(0.000000) → Tiempo (en ms) de la búsqueda de
```

```
                                                    orden (sin éxito) más lenta
      MaxUpdateStations(0.000000) → Tiempo (en ms) más lento en gestionar la
                                                    actualización de estaciones
      MaxTimeInDB(0.000000) → Tiempo (en ms) más en base de datos
  MaxPLCReadTime(1245.000000) → Tiempo máximo (en ms) de lectura de toda la
                                                    información de los PLCs
```

- ✓ Estadísticas del agente de averías →Activa las estadísticas referentes a tiempos, errores, contadores, etc. relativas al Agente que trata las averías. A continuación se detalla mediante un ejemplo cada uno de los campos que aparecen:

```
12:43:13.965 19 Mar 2010-
FailuresTransportAgent31::Statistics (in ms) → Entrada de la parte del Agente
==================================              de Transportes con sistema de
                                                control Galileo

WorkTime(0.000000)  SleepTime(4000.000000) → Tiempo en ejecutar las
                                              operaciones (en ms) / Tiempo de
                                              descanso
Current Cycle:
--------------
      Number of machines(1)     → Número de máquinas del computador
      Global failures time(0) → Tiempo (en ms) para gestionar las averías
                                globales
      Machine failures time(0) → Tiempo (en ms) para gestionar las averías de
                                 todas las máquinas
Stored data:
------------
      Global failures time stored(0)  → Tiempo máximo (en ms) para gestionar
                                        las averías globales
      Machine failures time stored(0) → Tiempo máximo (en ms) para gestionar
                                        las averías de todas las máquinas
```

- ✓ Solo en caso de éxito → Si se selecciona este check, las estadísticas relativas al Agente de Transportes solo se mostrarán en el caso de que suceda algo relevante, es decir, indicarán datos cuando realmente tenga sentido (en el caso por ejemplo de una búsqueda de orden se mostrará si se ha generado una orden pero no todos los ciclos que esté pidiendo una máquina infructuosamente)

2.2.3 Interacción entre Galileo y la periferia (Mapa de Memoria)

El diseño con el que se ha realizado Galileo está encaminado a intentar aislar la aplicación (en la mayor medida posible) de la dependencia a un fabricante de Hardware determinado. Para lograrlo Galileo delega todo el trato con el Hardware a una sería de módulos con un nombre similar a FBHAL.DLL (FieldBus Hardware Abstraction Layer). Mediante este mecanismo es posible realizar diferentes implementaciones de FBHAL.DLL para soportar diferentes fabricantes de hardware. Galileo utiliza internamente un mapa en memoria de la periferia y es con lo que trabaja. Este mapa se actualiza mediante el componente que reside en el módulo

de abstracción de hardware. Con lo cual el propio ejecutable Galileo no tiene contacto directo con hardware.

El sistema dispone de un área reservada de 32 Kbytes los cuales están repartidos de la siguiente manera:

Entradas	2048 Bytes
Shared Read	14336 Bytes
Salidas	2048 Bytes
Shared Write	14336 Bytes
Variables PLC	16384 Bytes

Por concepto el Sistema de Control Galileo prohíbe el direccionamiento absoluto de la periferia física, de forma que toda variable de acceso al hardware debe ser declarada. Esto que en principio puede parecer una falta de flexibilidad es una ventaja para lograr la rapidez en la programación. La única excepción a esta regla son las Variables PLC, que se han de mapear mediante direccionamiento absoluto obligatoriamente.

A priori se han definido en el sistema Galileo los siguientes tipos de datos:

- *PLC_BOOL*: Es un tipo de dato `booleano`, representa un `bit`, con lo cual puede tomar el valor de 0 o 1. Sin embargo por código se les debe asignar *`true`* o *`false`*. Ya que son los valores que pueden tener.
- *PLC_BYTE*: Este tipo de dato representa un `Byte` (8 Bits) y por lo tanto puede tener valores entre –128 y 127. Este tipo de dato es numérico.
- *PLC_WORD*: Representa un `word` (16 Bits), por lo tanto sus valores pueden estar comprendidos entre –32.768 y 32.767.
- *PLC_DWORD*: Representa un `dword` (32 Bits), tiene por tanto como valores posibles los comprendidos en el rango –2.147.483.648 a 2.147.483.647.
- *STRING*: Representa una cadena de texto estática. El que esta cadena de texto sea estática quiere decir que su contenido no puede variar durante la ejecución. El valor de esta cadena (valor que no podrá modificarse durante la ejecución) se asigna en el entorno de desarrollo *Designer*.
- *COMPONENT*: Representa un componente.

Para los tipos de datos PLC_BOOL, PLC_BYTE, PLC_WORD y PLC_DWORD se debe definir además un atributo que permita localizarlo en una de las diferentes zonas de memoria.

- Entradas**:** Esta zona de memoria está asociada a la periferia de entrada, por lo tanto se referirá la aplicación a cualquiera de los datos de entrada de la periferia distribuida.

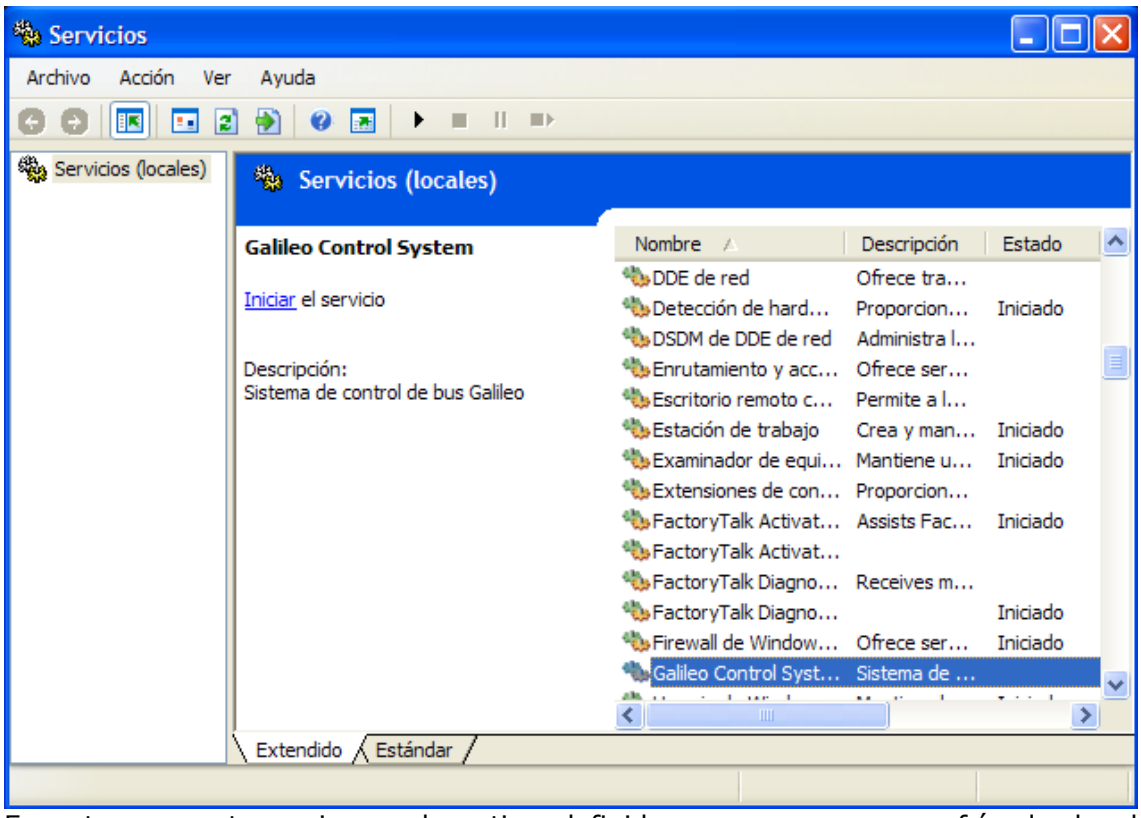
- Salidas**:** Esta zona de memoria está asociada a la periferia de salida, por lo tanto se referirá la aplicación a cualquiera de los datos de salida de la periferia distribuida.
- Shared Read**:** Esta zona de memoria está asociada a un área donde sólo se admiten escrituras externas al programa de control, es decir es una zona de memoria donde la aplicación de control no puede escribir, sin embargo puede declarar variables en esta zona a fin de recibir datos desde una aplicación externa (Visualización, etc.). Este tipo de variables está definido como remanentes.
- Shared Write**:** Esta zona de memoria está asociada a un área donde se permiten escrituras y lecturas por parte del programa de control. Es por tanto de dominio total del programa de control. Todas las variables definidas en esta zona son remanentes (es decir no se pierden ante una eventual caída del sistema de control).
- Variables PLC**:** Esta zona de memoria está asociada a la memoria de un PLC con el que se puede comunicar Galileo. Las variables que queramos visualizar de esta memoria externa deberán ser mapeadas mediante direccionamiento absoluto en esta área de memoria, indicando el tipo de memoria del PLC (Input, Output, Memory o DataBlock). En esta memoria sólo pueden mapearse variables de tipo Bool, Byte o Word.

2.2.4 Sistema de Arranque de la aplicación de control

De la misma forma que ocurre con los Autómatas Programables, la aplicación de control puede arrancar de dos modos diferentes:

2.2.4.1 Arranque Frío

El arranque en frío se produce cada vez que el servicio del sistema Galileo se inicia, esto es cada vez que el ordenador de control se reinicia, o cada vez que el servicio se reinicia desde el SCM (Service Control Manager)



En este momento se invoca la rutina definida como arranque en frío desde el entorno de desarrollo *Designer* para dicha computadora y posteriormente para cada máquina del programa se ejecuta la rutina de arranque frío definida en su secuenciador. Hay que hacer notar que en el momento de la ejecución de la rutina de arranque no existe aún periferia funcional (es decir no hay intercambio de Entradas / Salidas). Esta rutina debe ser utilizada para realizar las inicializaciones pertinentes, principalmente las cargas o configuraciones de componentes que requieran un elevado tiempo de ciclo (cargas de archivos de posiciones, cargas de archivos de configuración, etc.).

Nuevo en la versión III aparece el concepto de arranque por secuenciador de manera que todos los secuenciadores tienen la posibilidad de definir sus rutinas de arranque frío y caliente. De esta manera se aislará el código relativo al secuenciador no colocando ningún código necesario para el arranque de secuenciadores fuera del propio secuenciador en si mismo.

2.2.4.2 Arranque Caliente

El arranque caliente se produce cada vez que se para e inicia el programa de control desde el entorno de desarrollo Designer o desde la consola de control Galileo. En este momento la rutina que se invoca en el arranque es la definida como de arranque caliente y al igual que en el caso del arranque en frío, en el momento de la ejecución de la rutina no existe aún periferia funcional (es decir, no hay intercambio de Entradas / salidas).
Nuevo en la versión III aparece el concepto de arranque por secuenciador, de manera que todos los secuenciadores tienen la posibilidad de definir sus rutinas de arranque frío y caliente. De esta manera se aislará el código relativo al

secuenciador no colocando ningún código necesario para el arranque de secuenciadores fuera del propio secuenciador en si mismo.

2.2.5 Ejecución cíclica

Al igual que los PLC's el Sistema de Control Galileo soporta la programación Cíclica estructurada. De la misma manera que un S5 o S7 tiene un módulo de organización que se invoca cíclicamente (OB1) y desde el cual es posible invocar por programa cualquier otro módulo, en el caso de Galileo se puede definir cual es la función de ejecución cíclica y desde ésta invocar otras funciones definidas (al modo de programación estructurada).

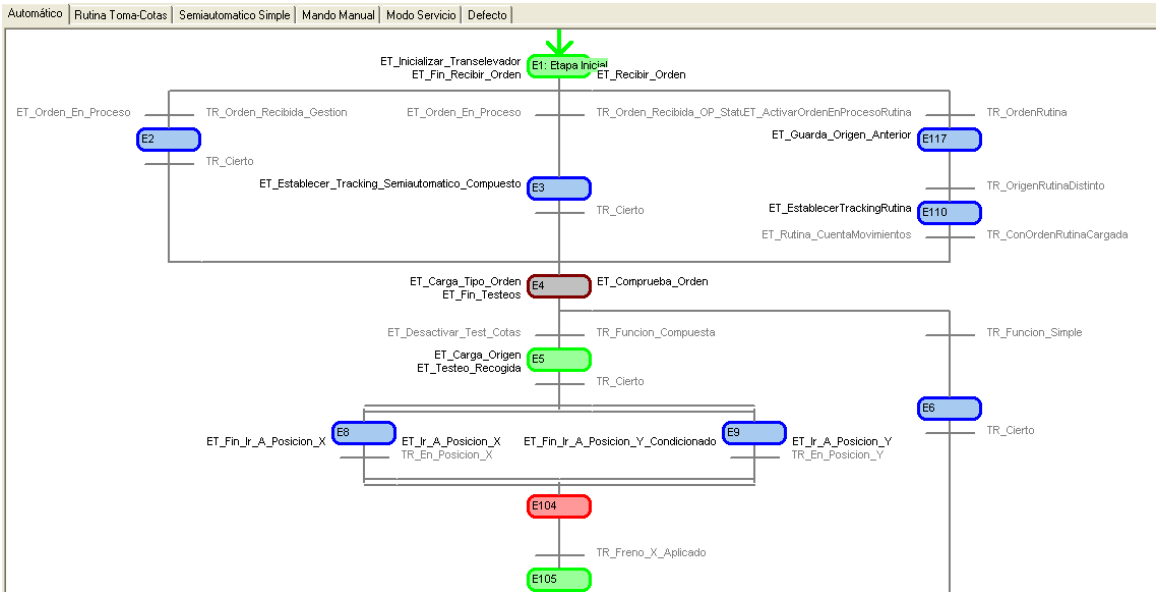
Hay que hacer notar que aunque se visualice un lenguaje de alto nivel, no debemos de dejar de pensar en que el sistema de ejecución es cíclico y que no se puede realizar la programación de un bucle sin fin dentro de una función de usuario (o el Sistema detendrá la transmisión de datos por perro guardián).

Pese a que este modo de funcionamiento puede ser utilizado para controlar una instalación (de la misma manera que se realiza con los PLC's) deja de tener sentido en el Sistema Galileo. El sistema elegido para desarrollar los programas, es un derivado de Grafcet, más evolucionado y que soporta un mecanismo similar a las excepciones. Este sistema de *Grafcet de Alto nivel* permite encapsular de forma conveniente el patrón de funcionamiento de una máquina de funcionamiento cíclico para aislarnos durante su programación de todos los factores que no sean su funcionamiento "teóricamente perfecto", posteriormente podemos definir las situaciones de fallo excepcionales y dotar al programa de lógica de comportamiento ante errores.

2.2.6 Funcionamiento del Grafcet de alto nivel

El funcionamiento lógico de los grafos de alto nivel será explicado en este apartado. Estos principios son básicos para el entendimiento de la lógica de programación, por lo cual, se recomienda prestar la mayor atención posible en este apartado.

Lo mejor es partir de un grafo para entender el funcionamiento general:



En la figura superior se puede ver parte de un grafo donde se aprecian los diferentes elementos característicos. A partir del mismo explicaremos el funcionamiento de la lógica de grafos.

Un grafo pretende describir y programar el funcionamiento de una máquina secuencial. A partir de esta premisa las secuencias se componen mediante etapas y transiciones.

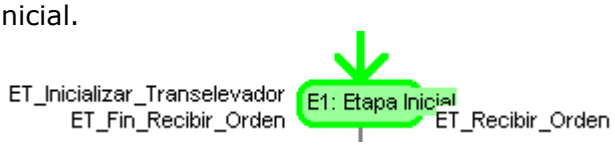
Como reglas generales (que se verán detalladas en el apartado de ejecución de un grafo) un grafo ejecuta una etapa activa hasta que es cierta su transición de salida, momento en el cual se activa la etapa siguiente, se desactiva la actual y prosigue la secuencia.

2.2.6.1 Etapas

Se define una etapa como un estado en el que se ejecuta una acción (o varias). Las etapas se ejecutan hasta que se dan las condiciones lógicas para avanzar. El *Sistema de Control Galileo* distingue entre varios tipos de etapas:

2.2.6.1.1 Etapas iniciales

Una etapa inicial es la que inicia la cadena secuencial, debe de existir una única etapa inicial y su existencia es obligatoria. La ejecución del grafo se iniciará en la etapa inicial (la primera vez que se arranque el programa).
Por ser una ejecución cíclica, una vez que el grafo llega a la etapa final vuelve a la inicial.



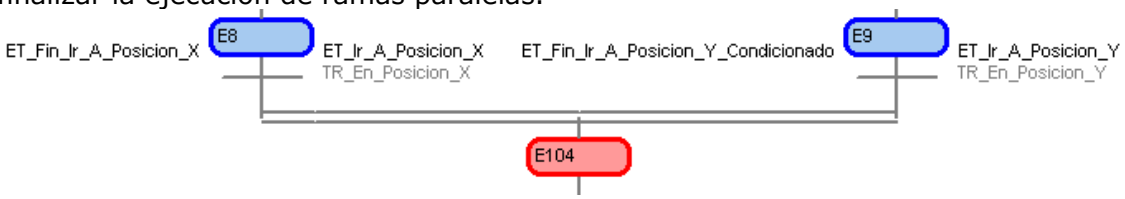
2.2.6.1.2 Etapas finales

Una etapa final es aquella que finaliza una cadena secuencial, debe existir una única etapa final y su existencia es obligatoria. La ejecución del grafo finaliza en la

etapa final, una vez que se cumple la transición de salida de esta etapa la ejecución retorna a la etapa inicial.

2.2.6.1.3 Etapas Wait

Una etapa Wait es aquella que fuerza al cumplimiento de todas las transiciones de entrada a la misma para poder validar su ejecución. Esta etapa se utiliza para finalizar la ejecución de ramas paralelas.



De forma independiente al tipo de etapa todas comparten una serie de atributos o propiedades que caracterizan su comportamiento en ejecución. Estos son:

2.2.6.1.4 Código de Error

El código de error indica qué código de error queremos notificar si se produce un timeout en la ejecución de la etapa (es decir si la etapa permanece más tiempo en ejecución del configurado).

NOTA: *Los errores de timeout no son errores que generen automáticamente una excepción; Son errores consultables por parte del programa usuario y con ellos se debe generar una excepción.*

2.2.6.1.5 Timeout

Indica el tiempo máximo que puede permanecer una etapa en ejecución. Si este tiempo se supera se notificará una avería (con el código de avería configurado en Código de Error).

2.2.6.1.6 Tiempo de Espera

Se define como tiempo de espera aquel a partir del inicio de la ejecución de una etapa en el cual no se evalúa la transición de salida, así que podemos afirmar que una vez configurado un tiempo de espera de pongamos por ejemplo 3000 mseg., la ejecución de la etapa va a durar al menos ese tiempo.

2.2.6.1.7 Acción de Entrada

Se define como acción de entrada a una función que no recibe ningún parámetro y no retorna ningún parámetro, esta acción se ejecutará una única vez por ciclo de grafo al producirse la activación de una etapa. Este tipo de acción suele servir para inicializar estados.

2.2.6.1.8 Acción Anterior

Se define como acción anterior aquella que se ejecuta antes de evaluar la transición de salida de una etapa, por tanto siempre se ejecuta al menos una vez. Esta acción es de ejecución cíclica, con lo cual mientras no se cumpla la transición de salida se continuará ejecutando.

2.2.6.1.9 Acción de Etapa

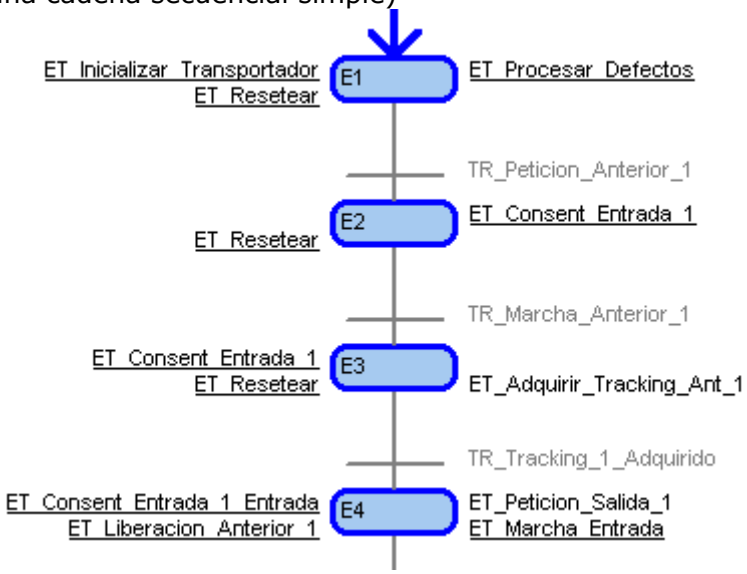
Se define como acción de etapa a aquella que se ejecuta de forma cíclica después de evaluar la transición de salida. Esto implica que si la transición es cierta ya al iniciarse la ejecución de una etapa, nunca se ejecutará la acción de etapa.

2.2.6.1.10 Acción de Salida

Se define como acción de salida a aquella que se ejecuta una única vez por ciclo de grafo cuando la etapa es desactivada (al cumplirse la(s) transición(es) de salida).

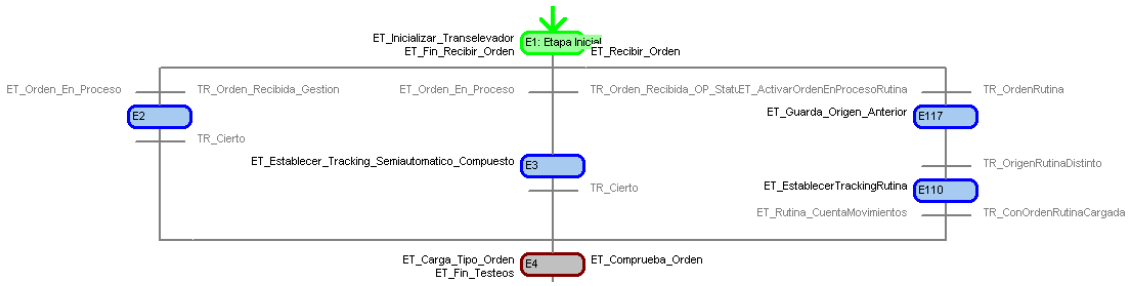
2.2.6.2 Transiciones

Se define como transición la condición de salida de una etapa o etapas. Representa la condición que se debe dar para pasar de una etapa a otra. Estas transiciones pueden estar conformando sistemas de ejecución ***normales*** (si la lógica de la aplicación es una cadena secuencial simple)



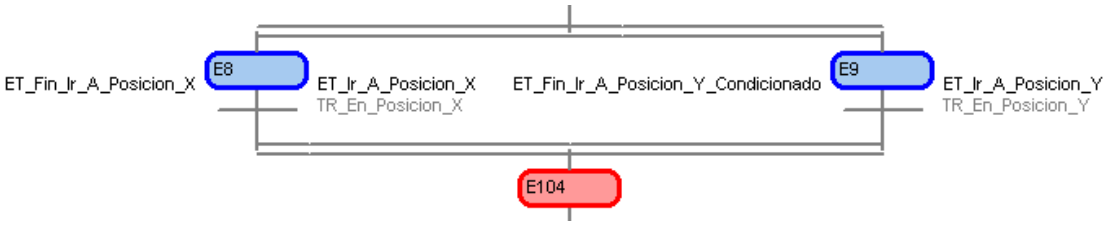
En este ejemplo la ejecución es muy simple ya que una vez que una etapa se activa se ejecuta hasta que la transición de salida es cierta. En este momento se desactiva y pasa a activarse la siguiente, y el ciclo continúa de la misma manera. Esta es la secuencia más simple posible.

a. Transiciones para ejecutar *ramas condicionales*:



En este caso *solamente se ejecutará una de las ramas en paralelo*, incluso aunque la transición de entrada de las tres sea cierta simultáneamente, en este caso se decidirá por orden de prioridad de manera que es una ejecución condicional exclusiva.
El hecho de que se ejecute *una y sólo una* de las líneas en paralelo sirve para evitar problemas lógicos de programación.

b. Transiciones para ejecutar *ramas en paralelo*:



En este caso se iniciarán las dos etapas en paralelo, las cuales continuarán su ejecución de forma independiente unas de otras. Una vez que cada línea paralela se finaliza se queda esperando hasta que todas las líneas paralelas llegan a la etapa Wait. Una vez finalizadas todas las etapas se activará la etapa wait y continuará el flujo.

Además del flujo de ejecución normal en toda instalación existen condiciones excepcionales que provocan que el sistema deba abandonar su normal ejecución para pasar a un estado y forma de actuar especial que permita controlar la situación de excepción.

La implementación de esta idea se logra en el Sistema de Control Galileo mediante un mecanismo de excepciones que se integra perfectamente en la lógica Grafcet. Este mecanismo de excepciones se dispara con lo que se denomina *Transiciones Any*. Una *transición any* es una transición que no va enlazada a dos etapas (como todas las demás) sino que se enlaza solamente con una etapa (a la salida de la etapa any). Representaría una condición que se evalúa siempre, no solamente cuando la etapa anterior está activa, de esta forma se interrumpirá la ejecución normal de Grafo para saltar a un estado de excepción si se cumple la transición any de entrada a esa excepción. El número de cadenas de control de excepción que se pueden definir está limitado a 255. Estas entradas disparan la ejecución de otro grafo del cual se saldría al cumplirse una transición de salida especial del tipo *transición return* (Aúnque no se haya terminado el flujo del GRAFO). De estar activa la etapa anterior a esta transición y cumplirse esta se retornaría el flujo de ejecución al grafo original en el punto donde se encontrase.

Mediante este mecanismo es posible aislar la ejecución normal de la secuencia de máquina del control de errores y modos de funcionamiento paralelos que se gestionaría mediante excepciones disparadas por *transiciones any* y finalizarían mediante *transiciones return*.

Por supuesto a estas transiciones se les puede asignar una prioridad de entrada para poder decidir si se cumplen varias transiciones any simultáneamente. De la misma manera se define que las *transiciones any* pueden interrumpir la ejecución de otra condición de excepción de forma anidada. Una *transición any* solamente puede interrumpir la ejecución cíclica u otra ejecución excepcional de menor prioridad.

En ningún caso se lleva un control tipo pila de las excepciones, es decir si una *transición any* de prioridad 1 interrumpe la ejecución normal y posteriormente otra de prioridad 2 interrumpe la ejecución de la excepción anterior cuando esta excepción retorne mediante su *transición return* lo hará al funcionamiento normal del grafo no a la primera excepción ocurrida. Aunque en el caso en que la condición para volver a la excepción de prioridad 1 persista volvería a entrar mediante esa Transición Any.

2.2.6.3 Acción de entrada de una transición

Las transiciones tienen la posibilidad de definir también una acción de entrada, que es una función que no recibe ningún parámetro y no retorna ningún parámetro, esta acción se ejecutará una única vez por ciclo de grafo al producirse la activación de una transición.

Este tipo de acción suele servir para inicializar estados.

2.2.6.4 Modos de funcionamiento

A partir de la versión 3.0 de Galileo, los grafos dejan de tener un sólo grafo por defecto, para pasar a soportar diferentes *Modos de Funcionamiento*. Un modo de funcionamiento no es más que un grafo simple, pero que ha de ser activado por el programador, de forma que ese grafo pasa a ser el grafo por defecto del secuenciador, y las etapas ANY retornan a una etapa del mismo, en lugar de al modo de funcionamiento anterior.

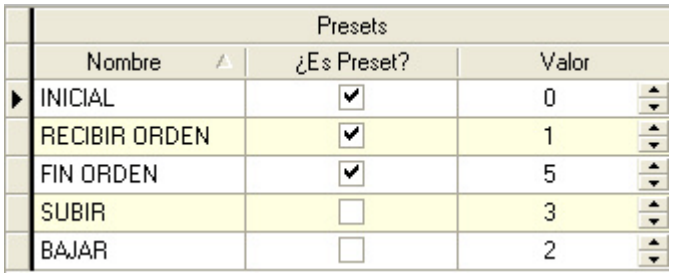
Se pueden definir hasta 256 modos de funcionamiento en cada secuenciador. El que tiene el identificador 0 es el que se toma como modo por defecto, de forma que siempre se inicializa a ese modo, hasta que el programa de control solicita el cambio. Las etapas ANY no cambian entre los distintos modos de funcionamiento.

2.2.7 Grafos PLC

Los grafos PLC son un tipo especial de grafos que se usan para describir las maquinas PLC al mismo nivel que lo hace Galileo, y que únicamente sirven como "envoltorio" a la lógica subyacente de dichas máquinas que en ningún momento es conocida por Galileo. A fin de que pueda ser posible la comunicación entre ambos sistemas (Galileo y Red de PLCs clásica), Galileo usa estos Grafos para describir unos conectores típicos que las máquinas ejecutándose en la red PLC deben implementar, tales como un espacio para almacenar trackings (con un formato

determinado), un área de memoria donde solicitar/indicar/recibir búsquedas de orden/PIEs/Fines de orden. Dichos secuenciadores no siguen el patrón de funcionamiento comentado para los anteriores Grafects de alto nivel, sino que simplemente actúan como etiquetas, no teniendo ninguna función descriptiva de su ejecución desde el punto de Galileo, salvo la referida de la búsqueda y finalización de orden o señalizar PIEs, en cuyo caso se comportan exactamente igual que las máquinas nativas de Galileo, dado que uno de los requisitos que se les exige a dichas máquinas es que sigan el mismo protocolo.

Existe, sin embargo, una manera de "etiquetar" las acciones que va realizando una máquina PLC, lo que nos permitirá conocer su estado, a la vez que interactuar con ella desde Galileo. Esto se consigue mediante la definición de Presets para el secuenciador de esas maquinas PLC:

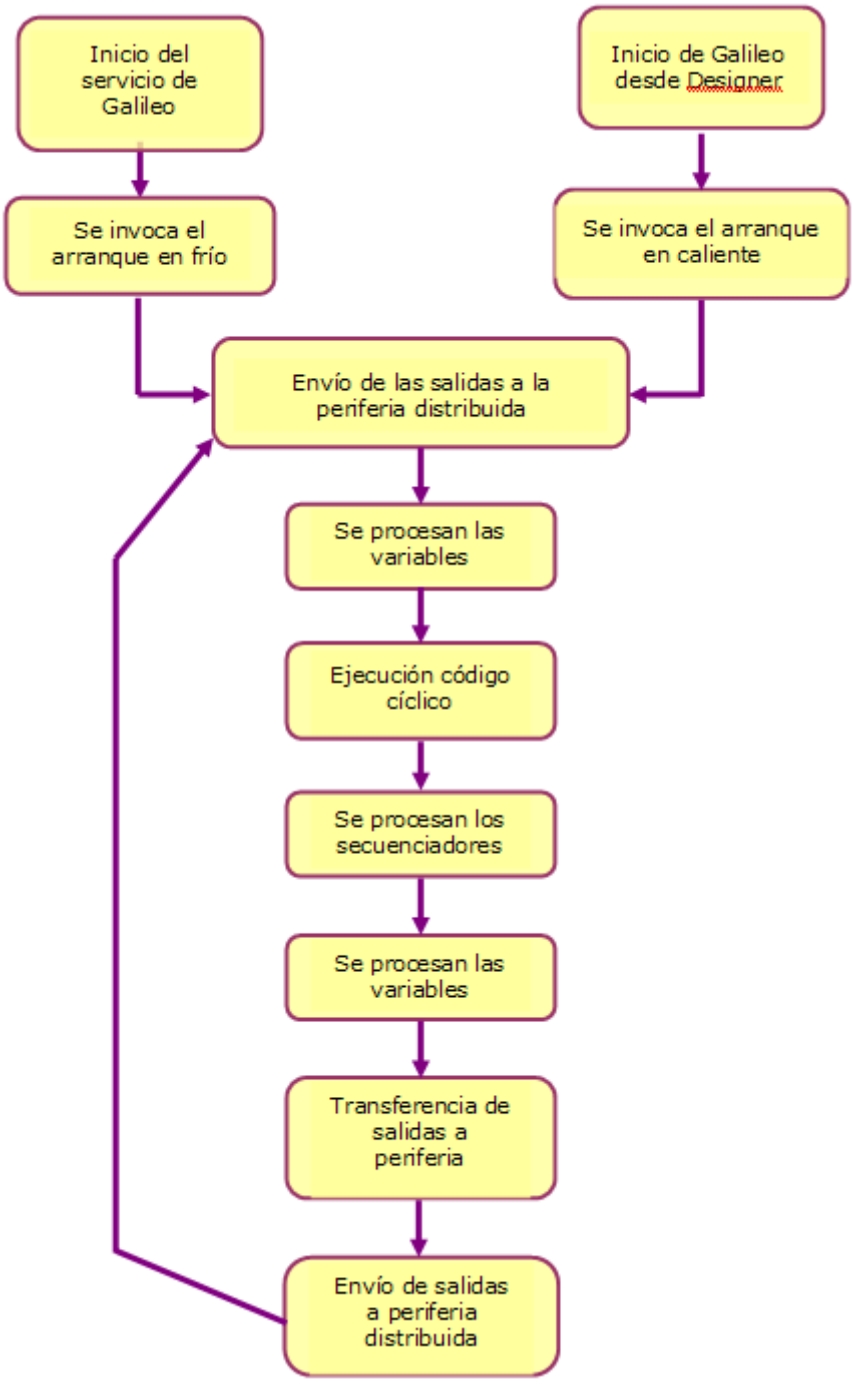


Estas entradas definen los posibles valores que se encuentran en una palabra que el PLC debe publicar a fin de poder pasar dicha información a Galileo, el cual podrá conocer su estado y modificarlo (colocando los valores marcados como "¿Es preset?" en dicha palabra). El secuenciador PLC también debe definir el número máximo de trackings que debe usar la máquina PLC, debiendo ser este número siempre menor que 10.

Los estados que no están marcados como "Preset", solo son indicativos, pero no se podrá requerir desde Galileo que la máquina PLC pase directamente a estos estados (forzar etapas).

2.2.8 Diagrama de ejecución de Galileo

El siguiente diagrama muestra el algoritmo de ejecución del Sistema Galileo. Mediante el mismo es posible analizar a más bajo nivel cual es la secuencia de ejecución.



Además se detallará el algoritmo de ejecución de un secuenciador para aclarar cómo se ejecuta el control del mismo.

En el caso de las variables de tipo PLC, hay una serie de anotaciones que realizar:

- No se procesan las salidas hacia la periferia, salvo que desde el Designer así se requiera explícitamente, es decir, el mapa en memoria solo es refrescado con los valores "reales" de la periferia, y solo en

casos puntuales se modifica desde Galileo. Esto es así debido a que Galileo actúa como un mero observador de dicha periferia, y es el código de control que llevan dichos PLCs el que realiza la activación/desactivación de dichas señales.

- La ejecución de los secuenciadores no realiza ninguna acción. Las modificaciones que Galileo puede/debe realizar van casi siempre relacionadas con el interfaz de comunicaciones con dichas máquinas (ordenes de transporte, tracking) ya que Galileo actúa para ellas como lo hacia anteriormente el Monitor de Transportes.

2.2.9 Extensibilidad del Sistema Galileo

El Sistema Galileo incorpora otro concepto clave en la programación moderna que es el de *Componente*. Un *Componente* es un trozo de código que incorpora una funcionalidad específica y que tiene sentido por sí mismo encapsulando un comportamiento. Este concepto permite definir componentes que realicen funciones complejas y aislarnos de las mismas mediante el interface que exponen. Una de las características más peculiares de los componentes en el Sistema Galileo es que son binarios, es decir, los componentes no son código Xana compilado a bytecodes y ejecutados por la máquina virtual. Un componente es un elemento que se ejecuta en el código binario nativo del procesador. En el caso del Sistema Galileo un componente reside en una dll (Dynamic Load Library) y ha sido escrito en un lenguaje de programación compilado (C++) de forma que su ejecución se realizará a la máxima velocidad permitida por el procesador.

De forma genérica podríamos dividir los componentes del sistema en 2 tipos:

2.2.9.1 Componentes de Encapsulación de Hardware

Este tipo de componentes encapsulan el manejo, utilización, configuración y programación de componentes de hardware utilizados frecuentemente en las instalaciones. El sistema Galileo por defecto incorpora componentes para un buen número de hardware utilizado en las instalaciones de almacenes y parkings automáticos, pero en todo caso puede ser extendido por los propios usuarios creando tantos componentes como necesiten.

2.2.9.2 Componentes de Lógica de Sistema

Este tipo de componentes sería el que no tiene una interrelación directa con el hardware de un elemento en particular sino que tiene que ver con el propio hardware del sistema o bien con elementos lógicos que se encuentran en numerosas instalaciones. El sistema Galileo incorpora de serie los componentes claves, pero nadie impide el desarrollo futuro de más funcionalidades.

3 SISTEMA DE CONTROL GALILEO: LENGUAJE DE PROGRAMACIÓN XANA

El lenguaje de programación utilizado a lo largo del sistema Galileo se denomina XANA, es un derivado del C/C++ recortado en la mayoría de sus características y ampliado en otras. Este lenguaje se estructura atendiendo a las necesidades básicas de la programación de control con lo cual es un lenguaje que elimina la mayoría de los puntos oscuros del C para limitar la posibilidad de cometer errores.

Este lenguaje es interpretado, lo que significa que es más lento que un lenguaje compilado, pero tiene a su favor el hecho de que permite una más fácil depuración y que los servicios de depurado que se pueden implementar son mucho mayores.

Este lenguaje de programación se compila a un sistema de bytecodes que son posteriormente interpretados por la máquina virtual. Para acelerar la ejecución y favorecer la normalización, dispone del concepto de componente. Un componente es un objeto nativo (es decir programado en C++ y compilado) que "vive" dentro de una DLL (Librería de Enlace Dinámico). En una DLL puede existir uno o más componentes, sin límite a la cantidad de los mismos que se pueden ubicar dentro de la DLL.

3.1 Conceptos básicos

Xana es un lenguaje procedural de nivel medio diseñado para la programación de secuenciadores. Por regla general trata de seguir la línea definida por C, de tal manera que salvo excepciones muy localizadas *Xana* puede ser considerado como un subconjunto bastante mínimo de C; esta gran semejanza con un lenguaje de programación tan extendido como C hace que la curva de aprendizaje tienda a ser bastante menor.

Dado lo específico del ámbito de aplicación de *Xana*, el lenguaje presenta ciertas características que no se encuentran en C, aún cuando estas son mínimas y tratan de seguir al máximo la definición de C, o en su defecto de C++. Para facilitar el aprendizaje, las características del lenguaje no presentes en C se indicarán como tales.

Un fichero de código fuente *Xana* se compila con el compilador *xanac*, produciéndose un fichero denominado *módulo*, con extensión .tbc, este módulo se procesa por el entorno de desarrollo uniéndole información de variables y secuenciadores, se comprime y se genera con todo ello un archivo de extensión .plc que podrá ser ejecutado en el sistema Galileo. Un módulo .tbc no tiene sentido sin el resto de información que añade al archivo .plc.

3.2 Elementos del lenguaje

Xana es un lenguaje de formato libre, de tal manera que el significado del código fuente no se ve afectado por los espacios en blanco, tabuladores y / o saltos de línea presentes entre los elementos del lenguaje. Para ser precisos, varios espacios

en blanco, tabuladores y / o saltos de línea seguidos son equivalentes a un espacio en blanco.

3.2.1 Comentarios

Xana admite el uso de comentarios al estilo *C++*, es decir, comentarios de una sola línea, comenzando por *//.* Los comentarios pueden colocarse en cualquier lugar del código fuente donde un espacio en blanco sea legal, teniendo en cuanta que ha de quedar separado por al menos un espacio de otros elementos:

```
// esto es un comentario válido en Xana, al igual
// que lo sería en C++
```

Xana no soporta el estilo de comentario de bloque de C, es decir, comentarios entre /* y */.

3.2.2 Tipos y variables

En el Sistema de Control Galileo existen los siguientes tipos de datos:

- *PLC_BOOL*: Es un tipo de dato `booleano`, representa un `bit`, con lo cual puede tomar el valor de 0 o 1. Sin embargo por código se les debe asignar *`true`* o *`false`*. Ya que son los valores que pueden tener.
- *PLC_BYTE*: Este tipo de dato representa un `Byte` (8 Bits) y por lo tanto puede tener valores entre –128 y 127. Este tipo de dato es numérico.
- *PLC_WORD*: Representa un `word` (16 Bits), por lo tanto sus valores pueden estar comprendidos entre –32.768 y 32.767.
- *PLC_DWORD*: Representa un `dword` (32 Bits), tiene por tanto como valores posibles los comprendidos en el rango –2.147.483.648 a 2.147.483.647.
- *PLC_QWORD*: Representa un `qword` (64 Bits), tiene por tanto como valores posibles los comprendidos en el rango - 9223372036854775807 a 9223372036854775807.
- *PLC_DOUBLE*: Representa una `double` (64 bits), es decir un valor flotante.
- *STRING*: Representa una cadena de texto estática. El que esta cadena de texto sea estática quiere decir que su contenido no puede variar durante la ejecución. El valor de esta cadena (valor que no podrá modificarse durante la ejecución) se asigna en el entorno de desarrollo *Designer*.
- *COMPONENT*: Representa un componente.

Para los tipos de datos PLC_BOOL, PLC_BYTE, PLC_WORD y PLC_DWORD se debe definir además un atributo que permita localizarlo en una de las diferentes zonas de memoria.

- Entradas**:** Esta zona de memoria está asociada a la periferia de entrada, por lo tanto se referirá la aplicación a cualquiera de los datos de entrada de la periferia distribuida.
- Salidas**:** Esta zona de memoria está asociada a la periferia de salida, por lo tanto se referirá la aplicación a cualquiera de los datos de salida de la periferia distribuida.
- Shared Read**:** Esta zona de memoria está asociada a un área donde sólo se admiten escrituras externas al programa de control, es decir es una zona de memoria donde la aplicación de control no puede escribir, sin embargo puede declarar variables en esta zona a fin de recibir datos desde una aplicación externa (Visualización, etc.). Este tipo de variables está definido como remanentes.
- Shared Write**:** Esta zona de memoria está asociada a un área donde se permiten escrituras y lecturas por parte del programa de control. Es por tanto de dominio total del programa de control. Todas las variables definidas en esta zona son remanentes (es decir no se pierden ante una eventual caída del sistema de control).
- Variables PLC**:** Esta zona de memoria está asociada a la memoria de un PLC con el que se puede comunicar Galileo. Las variables que queramos visualizar de esta memoria externa deberán ser mapeadas mediante direccionamiento absoluto en esta área de memoria, indicando el tipo de memoria del PLC (Input, Output, Memory o DataBlock). En esta memoria sólo pueden mapearse variables de tipo Bool, Byte o Word.

Toda información a manejar en un programa escrito en Xana ha de estar almacenada en variables, las cuales han de ser declaradas previamente a su uso. La definición de una variable conlleva varias cosas:

✓ Asignación de un nombre simbólico a un elemento de información en memoria, denominado *identificador*. Ejem:

```
E_ParoAdelante
```

✓ Determinación del tipo del elemento, lo que especifica cómo se tratará ésta (no es lo mismo tratar un carácter que un número entero). Ejem:

```
Nombre: E_ParoAdelante
Estación de Bus: BM_57
Input/Output: Input
Tamaño de la variable: BIT
Byte origen: 0
```

Los caracteres permitidos para el identificador son:

✓ *Caracteres numéricos* (‘0’ a ‘9’), excepto el caso del primer carácter.

- ✓ *Caracteres alfabéticos* ('a' a 'z' y 'A' a 'Z'). No se admiten caracteres no internacionales, como vocales acentuadas o la 'ñ'.
- ✓ El carácter '_'

Así pues, identificadores válidos pueden ser:

```
Qwerty  a  a34  ttttttttt cnt
```

No hay limitación en el tamaño de los identificadores. Por el contrario, no serían válidos los siguientes identificadores:

```
1a2b3c a-b-d una.vez 123 $33
```

Xana reserva para sí una serie de identificadores que no pueden ser usados para variables. Estos se denominan palabras reservadas, y tienen asignadas funciones del lenguaje que se irán comentando poco a poco. La lista completa de palabras reservadas puede verse en la sección 5.3, Tabla de palabras reservadas. Es necesario hacer notar que los identificadores son sensibles a mayúsculas; de tal manera que *uno* y *Uno* son diferentes identificadores; de igual forma, *Byte* es un identificador válido, mientras que *byte* no lo es por ser palabra reservada.
La declaración de una variable se realiza de la siguiente manera:

```
<tipo> <identificador> ;
```

Donde *<tipo>* será un texto que define el tipo de la variable, e *<identificador>* será el nombre o identificador con el que se conocerá la variable.
El tipo puede ser uno de los siguientes:

- *byte*: la variable se referirá a un byte en memoria. Se interpreta como un entero con signo, en el rango [-128, 127], almacenado en complemento a 2.
- *word*: la variable se referirá a una palabra (2 `bytes`) contiguos en memoria. Se interpreta como un entero con signo en el rango [-32768, 32767], almacenado en complemento a 2.
- *dword*: la variable se referirá a una doble palabra (4 `bytes`) contiguas en memoria. Se interpreta como un entero con signo en el rango [-2147483648, 2147483647], almacenado a complemento a 2.
- *bool*: la variable encapsula un valor booleano, cuyos valores posible son `true` y `false`
- *double*: la variable se referirá a una doble palabra (4 `bytes`) contiguas en memoria. Se interpreta como un numero real (flotante) con signo en el rango [$3.4 * 10^{-38}$,$3.4 * 10^{+38}$].
- *long*: la variable se referirá a una quadruple palabra (8 `bytes`) contiguas en memoria. Se interpreta como un entero con signo en el rango [-2^{63}, $2^{63} - 1$], almacenado a complemento a 2.

Son declaraciones válidas las siguientes:

```
byte unByte;      // define una variable de 1 byte
byte otro_byte;   // define una variable de 1 byte
dword unaDPalabra; // define una variable de 4 bytes
```

```
double unflotante;// define una variable de 4 bytes de tipo real
Machine seq1;     // define una variable de tipo Seq
                  // (objeto externo)
```

3.2.2.1 Modificadores

Además del tipo y del identificador, una variable puede declararse especificando una serie de modificadores, que especifican otros aspectos de la variable.

Actualmente existe un conjunto de modificadores que especifican cómo se almacenará la variable en memoria. Estos modificadores son palabras reservadas que se anteponen al tipo en la declaración:

- *input*: la variable estará ligada al bus de campo, y será considerada una variable de entrada.
- *output*: la variable estará ligada al bus de campo, y será considerada una variable de salida.
- *shared*: la variable estará ubicada en memoria compartida.
- *persistent*: la variable estará ubicada en memoria no volátil.

En la declaración de una variable se admite especificar uno de estos modificadores, o bien ninguno, en cuyo caso se sobreentiende el uso de *shared*.

Xana no define, ni permite definir, la ubicación física exacta de las variables esta tarea se realiza en el entorno de programación *Designer* y por lo tanto toda la información anterior sirve para entender el sistema no para modificar el código generado por *Designer*.

Debe hacerse notar también que la aplicación *Designer* mantiene la información de tipo y esta es la que impera en la ejecución. Si un programa es alterado después de generado para cambiar modificadores de tipo se estaría perdiendo el tiempo ya que la información de tipo que prevalece es la originaria de Designer**.**

3.2.2.2 Constantes

Una constante es un valor conocido e inmutable, es decir, que no cambia. *Xana* permite usar valores constantes, pero no permite su definición simbólica (esto es, asociarle un nombre) como hace *C* o *C++. Xana* permite el uso de dos tipos de constantes:

1. *Constantes numéricas*: son de tipo *dword* o *long* y por tanto equivalen a enteros con signo en el rango [-2^{63}, 2^{63} - 1]. Pueden escribirse de tres maneras diferentes:
   a. *Notación decimal*: uno o varios dígitos seguidos, precedidos o no por un signo '–' o '+'

      ```
      -12345
      ```

   b. *Notación hexadecimal*: los caracteres '0x' seguidos de uno o varios dígitos hexadecimales (dígitos del 0 al 9 y las letras de la 'a' a la 'f', minúsculas o mayúsculas), todo ello precedido o no por un signo '–' o '+'

```
0xF1E
```

c. *Notación binaria*: uno o más dígitos binarios seguidos de la letra 'b'

```
011100b
```

Para el caso de valores fuera del rango [-2147483648, 2147483647], solo se admite el formato decimal.
No se realiza ningún tipo de comprobación sobre el tamaño de la constante. Si se utiliza una constante de valor fuera del rango admitido se producirá un desbordamiento, quedando el valor efectivo de tal constante indefinido (ver sección 5.2.2.12, Coerciones y desbordamientos).

2. *Constantes booleanas*: son de tipo `bool`, y pueden tomar por tanto dos valores: `true` y `false`. Por tanto, existen sólo dos constantes válidas, especificadas por las cadenas de texto 'true' y 'false'.
3. *Constantes flotantes*: son de tipo `double` y su formato puede ser tanto un numero decimal (con punto decimal) o en notación científica.

3.2.2.3 Expresiones y operadores

Una vez que queda explicado como definir variables y como especificar constantes, el siguiente paso es explicar que se puede hacer con ellas. En el contexto de un lenguaje imperativo como Xana (o C), la potencia viene dada por lo que se puede hacer con estas entidades.

Xana permite realizar operaciones aritméticas, operaciones a nivel de bit, operaciones de comparación y operaciones booleanas[1] con las variables y constantes definidas. El resultado de estas operaciones puede a su vez ser utilizado en otra operación, o bien almacenado en una variable. Es importante resaltar que para poder utilizar una variable ésta ha de ser previamente definida.
Cada una de estas operaciones tiene asignado un operador, el cual toma una serie de operandos (uno o dos) y devuelve un valor. Tómese como ejemplo la operación *suma*, denotada por el operador +: Toma dos operandos, y retorna como valor la suma de sus operandos:

```
var1 + var2 // operación que retorna la suma de var1 y var2
```

El conjunto de un operador y sus operandos se denomina expresión. Una expresión tiene un valor (el resultado de la operación especificada por el operador sobre los operandos) y, al igual que las variables, también tiene un tipo. A todos los efectos, una expresión puede ser utilizada donde una variable de su mismo tipo esté permitida, de tal manera que uno puede construir expresiones con otras expresiones como operandos. Esto refleja el concepto matemático de expresión, donde uno puede concatenar sumas con multiplicaciones y otras operaciones para obtener un resultado global.

```
var1 + var2 + var3 / var4  // expresión compleja formada
                           // por varias subexpresiones
```

[1] Operaciones de álgebra de Boole, es decir, OR, AND, NOT y XOR

3.2.2.4 Expresiones y tipos

Como queda dicho, una expresión tiene asociado un tipo, al igual que una variable o una constante. Los tipos posibles para una expresión son los mismos que para una variable (`byte`, `word`, `dword`, `bool` u objeto externo). Más explícitamente, el tipo de una expresión depende del tipo de sus operandos y del operador concreto utilizado. Es mas, para determinadas combinaciones de tipos de operandos y operadores el tipo de la expresión no está definido, si es que la operación tampoco lo está. Este es el caso de la suma de un `dword` con un `booleano`, o la multiplicación de un byte por un objeto externo. En caso de que una expresión produzca un tipo no definido se asume que la expresión no es correcta, y por tanto el compilador fallará emitiendo un mensaje de error explicativo.

El sistema de tipos de *Xana* es bastante más restrictivo que el de *C*; *C* permite aplicar operadores de predicado a variables enteras, o usar enteros como predicados. Esto permite hacer código más flexible, pero también permite cometer errores que pasan desapercibidos a los ojos del programador novel. Considerando que en el contexto de aplicación de *Xana* el uso de un sistema de tipos más permisivo no conduciría mas que a lo segundo, sin aportar ventajas significativas, se opta por restringir el sistema de tipos[2].

A continuación se describen todos los operadores soportados por *Xana*, describiéndose las combinaciones de tipos válidos y su correspondiente tipo de retorno.

3.2.2.5 Operadores aritméticos

Son operadores que definen operaciones matemáticas. Se definen sólo las operaciones básicas; operaciones como exponenciación, potencia, y operadores trigonométricos se consideran fuera del ámbito de aplicación de *Xana*, y por tanto no se implementan[3].

- Operador **+**

Semántica	Suma dos valores
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
`byte`	`byte`	`byte`
`byte`	`word`	`word`
`byte`	`dword`	`dword`
`byte`	`long`	`long`
`byte`	`double`	`double`
`word`	`byte`	`word`

[2] Para aquellos que conozcan *C* y *C++*, puede añadirse que la adición de un tipo boolean permite restringir el sistema de tipos. El sistema de tipos de *C++* no fue restringido por mantener la compatibilidad hacia atrás.
[3] No obstante, podrían ser implementados mediante el uso de objetos externos o funciones. Ambos conceptos se verán en detalle más adelante

word	word	word
word	dword	dword
word	long	long
word	double	double
dword	byte	dword
dword	word	dword
dword	dword	dword
dword	long	long
dword	double	double
double	byte	double
double	word	double
double	dword	double
double	long	double
double	double	double
long	byte	long
long	word	long
long	dword	long
long	long	long
long	double	double

- Operador **–**

Semántica	Sustrae un valor de otro
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
byte	byte	byte
byte	word	word
byte	dword	dword
byte	long	long
byte	double	double
word	byte	word
word	word	word
word	dword	dword
word	long	long
word	double	double
dword	byte	dword
dword	word	dword
dword	dword	dword
dword	long	long
dword	double	double
double	byte	double
double	word	double
double	dword	double
double	long	double
double	double	double
long	byte	long
long	word	long
long	dword	long
long	long	long
long	double	double

- Operador *****

Semántica	Multiplica dos valores
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
byte	byte	byte
byte	word	word
byte	dword	dword
byte	long	long
byte	double	double
word	byte	word
word	word	word
word	dword	dword
word	long	long
word	double	double
dword	byte	dword
dword	word	dword
dword	dword	dword
dword	long	long
dword	double	double
double	byte	double
double	word	double
double	dword	double
double	long	double
double	double	double
long	byte	long
long	word	long
long	dword	long
long	double	double
long	long	long

- Operador **/**

Semántica	División entera de dos valores. El resultado es un número entero, que se identifica con el cociente entero de la división del primer operando por el segundo. Por ejemplo, 17 / 3 produce el valor 5. La excepción a esta regla es cuando uno de los operadores es de tipo `double`, en cuyo caso el resultado es un número en notación decimal.
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
byte	byte	byte
byte	word	word
byte	dword	dword
byte	long	long

byte	double	double
word	byte	word
word	word	word
word	dword	dword
word	long	long
word	double	double
dword	byte	dword
dword	word	dword
dword	dword	dword
dword	long	long
dword	double	double
double	byte	double
double	word	double
double	dword	double
double	long	double
double	double	double
long	byte	long
long	word	long
long	dword	long
long	double	double
long	long	long

- Operador **%**

Semántica	Calcula el resto de la división entera del primer operando por el segundo. Por ejemplo, 17 % 3 = 2. En el ámbito de C esta operación se denomina *módulo*.
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
byte	byte	byte
byte	word	word
byte	dword	dword
byte	long	long
word	byte	word
word	word	word
word	dword	dword
word	long	long
dword	byte	dword
dword	word	dword
dword	dword	dword
dword	long	long
long	byte	long
long	word	long
long	dword	long
long	long	long

- Operador **++**

Semántica	Incrementa en 1 el valor de una variable. Al contrario que en C la expresión resultante no produce resultado (es decir, no es usable como expresión), ni se puede aplicar sobre algo que no sea una variable (no se puede aplicar sobre otra expresión)
	Esto implica que el único uso de este operador es su aplicación directa sobre una variable, sin que el resultado se use para nada puesto que no está definido:
	
	`dword d;`
	`d = 3;`
	`d++;  // d vale ahora 4`
	`d = d++;  // error, d++ no es usable como expresión`
	`(d + 1)++; // error, no se puede incrementar lo que`
	`// no sea una variable`

Nº operadores	1

Tabla de tipos permitidos

Operando 1	Resultado
byte	void
word	void
dword	void
long	void

- Operador −

Semántica	Decremento en 1 el valor de una variable. Al contrario que en C la expresión resultante no produce resultado (es decir, no es usable como expresión), ni se puede aplicar sobre algo que no sea una variable (no se puede aplicar sobre otra expresión)
	Esto implica que el único uso de este operador es su aplicación directa sobre una variable, sin que el resultado se use para nada puesto que no está definido:
	
	`dword d;`
	`d = 3;`
	`d--; // d vale ahora 2`
	`d = d--; // error, d-- no es usable como expresión`
	`(d + 1)--; // error, no se puede decrementar algo que`
	`// no sea una variable`

Nº operadores	1

Tabla de tipos permitidos

Operando 1	Resultado
byte	void
word	void
dword	void
long	void
double	void

3.2.2.6 Operadores de comparación

Estos operandos permiten construir predicados; se entiende por predicado una expresión de tipo booleano, o sea, una expresión cuyos valores posibles son *true* y *false*, o *verdadero* y *falso*.

- Operador **<**

Semántica	Compara dos valores. Retorna `true` si el primero es estrictamente[4] menor que el segundo, `false` en otro caso
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
byte	byte	bool
byte	word	bool
byte	dword	bool
byte	long	bool
byte	double	bool
word	byte	bool
word	word	bool
word	dword	bool
word	long	bool
word	double	bool
dword	byte	bool
dword	word	bool
dword	dword	bool
dword	long	bool
dword	double	bool
double	byte	bool
double	word	bool
double	dword	bool
double	long	bool
double	double	bool

- Operador **>**

Semántica	Compara dos valores. Retorna `true` si el primero es estrictamente mayor que el segundo, `false` en otro caso
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
byte	byte	bool

[4] Matemáticamente la palabra menor ya implica que el menor es estricto. Nos permitimos la redundancia a fin de recalcar el concepto.

byte	word	bool
byte	dword	bool
byte	long	bool
byte	double	bool
word	byte	bool
word	word	bool
word	dword	bool
word	long	bool
word	double	bool
dword	byte	bool
dword	word	bool
dword	dword	bool
dword	long	bool
dword	double	bool
double	byte	bool
double	word	bool
double	dword	bool
double	long	bool
double	double	bool

- Operador **>=**

Semántica	Compara dos valores. Retorna *true* si el primero es mayor o igual que el segundo, *false* en otro caso
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
byte	byte	bool
byte	word	bool
byte	dword	bool
byte	long	bool
byte	double	bool
word	byte	bool
word	word	bool
word	dword	bool
word	long	bool
word	double	bool
dword	byte	bool
dword	word	bool
dword	dword	bool
dword	long	bool
dword	double	bool
double	byte	bool
double	word	bool
double	dword	bool
double	long	bool
double	double	bool

- Operador **<=**

Semántica	Compara dos valores. Retorna *true* si el primero es menor o igual que el segundo, *false* en otro caso
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
byte	byte	bool
byte	word	bool
byte	dword	bool
byte	long	bool
byte	double	bool
word	byte	bool
word	word	bool
word	dword	bool
word	long	bool
word	double	bool
dword	byte	bool
dword	word	bool
dword	dword	bool
dword	long	bool
dword	double	bool
double	byte	bool
double	word	bool
double	dword	bool
double	long	bool
double	double	bool

- Operador ==

Semántica	Compara dos valores. Retorna *true* si el primero es igual al segundo, *false* en otro caso
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
bool	bool	bool
byte	byte	bool
byte	word	bool
byte	dword	bool
byte	long	bool
byte	double	bool
word	byte	bool
word	word	bool
word	dword	bool
word	long	bool
word	double	bool
dword	byte	bool
dword	word	bool
dword	dword	bool

dword	long	bool
dword	double	bool
double	byte	bool
double	word	bool
double	dword	bool
double	long	bool
double	double	bool

3.2.2.7 Operadores de predicado

Estos operadores permiten construir predicados complejos a partir de predicados simples. Implementan los operadores definidos por el álgebra de Boole: *AND*, *OR*, y *NOT*. Como añadidura se implementa la operación *XOR*.

Todos estos operandos operan exclusivamente sobre predicados, y producen a su vez otro predicado. Por ello, el único tipo admitido para los operandos es `bool` (es decir, los valores posibles de los operandos serán `true` y `false`), y el tipo de la expresión resultante siempre será `bool`[5].

- Operador **&&**

Semántica	Retorna `true` si ambos operandos son `true`; retorna `false` en otro caso
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
bool	bool	bool

- Operador **||**

Semántica	Retorna `true` si al menos uno de los operandos es `true`; retorna `false` en otro caso
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
bool	bool	bool

- Operador **!**

Semántica	Retorna `true` si el operando es `false`; retorna `false` en otro

[5] Esto representa una gran restricción con respecto a C, donde el tipo de los operandos sólo precisa ser escalar (carácter o entero)

	caso
Nº operadores	1

Tabla de tipos permitidos

Operando	Resultado
bool	bool

3.2.2.8 Operadores a nivel de bit

Estos operadores permiten realizar manipulaciones a nivel de bit dentro de las variables. Permiten por tanto comprobar y / o cambiar bits de una palabra de manera individual. Las variables de tipo *double* no permiten usar estos operadores.

- Operador **&**

Semántica	Retorna el resultado de realizar un AND a nivel de bit sobre los dos operandos.
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
byte	byte	byte
byte	word	word
byte	dword	dword
byte	long	long
word	byte	word
word	word	word
word	dword	dword
word	long	long
dword	byte	dword
dword	word	dword
dword	dword	dword
dword	long	long
long	byte	long
long	word	long
long	dword	long
long	long	long

- Operador **|**

Semántica	Retorna el resultado de realizar un OR a nivel de bit sobre los dos operandos.
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
byte	byte	byte
byte	word	word

byte	dword	dword
byte	long	long
word	byte	word
word	word	word
word	dword	dword
word	long	long
dword	byte	dword
dword	word	dword
dword	dword	dword
dword	long	long
long	byte	long
long	word	long
long	dword	long
long	long	long

- Operador ^

Semántica	Retorna el resultado de realizar un XOR (OR exclusivo) a nivel de bit sobre los dos operandos.
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
byte	byte	byte
byte	word	word
byte	dword	dword
byte	long	long
word	byte	word
word	word	word
word	dword	dword
word	long	long
dword	byte	dword
dword	word	dword
dword	dword	dword
dword	long	long
long	byte	long
long	word	long
long	dword	long
long	long	long

- Operador ~

Semántica	Retorna el resultado de realizar una negación a nivel de bit sobre el operando.
Nº operadores	2

Tabla de tipos permitidos

Operando	Resultado
byte	byte

word	word
dword	dword
long	long

- Operador **<<**

Semántica	Realiza un desplazamiento a la izquierda de los bits del primer operando, tantas posiciones como se indica en el segundo operando, conservando el signo.
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
byte	byte	byte
byte	word	byte
byte	dword	byte
byte	long	byte
word	byte	word
word	word	word
word	dword	word
word	long	word
dword	byte	dword
dword	word	dword
dword	dword	dword
dword	long	dword
long	byte	long
long	word	long
long	dword	long
long	long	long

- Operador **>>**

Semántica	Realiza un desplazamiento a la derecha de los bits del primer operando, tantas posiciones como se indica en el segundo operando, conservando el signo.
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
byte	byte	byte
byte	word	byte
byte	dword	byte
byte	long	byte
word	byte	word
word	word	word
word	dword	word
word	long	word
dword	byte	dword
dword	word	dword

dword	dword	dword
dword	long	dword
long	byte	long
long	word	long
long	dword	long
long	long	long

- Operadores **<>** y **!=**

Semántica	Comparan dos valores. Retornan *true* si el primero es distinto al segundo, *false* en otro caso. Ambos operadores son equivalentes, presentando estas dos formas a imagen y semejanza de *C*.
Nº operadores	2

Tabla de tipos permitidos

Operando 1	Operando 2	Resultado
bool	bool	bool
byte	byte	bool
byte	word	bool
byte	dword	bool
byte	long	bool
byte	double	bool
word	byte	bool
word	word	bool
word	dword	bool
word	long	bool
word	double	bool
dword	byte	bool
dword	word	bool
dword	dword	bool
dword	long	bool
dword	double	bool
double	byte	bool
double	word	bool
double	dword	bool
double	long	bool
double	double	bool

3.2.2.9 Asignación

Mediante el uso de variables y constantes, combinándolos entre sí con operadores, podemos realizar cálculos de cierta complejidad. Pero lo más seguro que sea necesario almacenar el resultado de tales cálculos en algún sitio para su uso futuro.

Xana permite la asignación de resultados a variables, siempre y cuando los tipos de la expresión y de la variable sean iguales, o al menos compatibles (ver sección *Coerciones y desbordamientos*).

```
// declaramos tres variables de tipo byte
byte b1;
byte b2;
byte res;
// sumamos las dos primeras y dejamos el resultado en la tercera
res = b1 + b2;
```

Apréciese que toda asignación ha de terminar con punto y coma. Se pueden realizar asignaciones un tanto sorprendentes a primera vista:

```
b1 = b1 + 1;
```

Esta operación, que matemáticamente es incorrecta, está permitida en casi cualquier lenguaje de programación. El truco reside en que el '=' no denota *igualdad*, como en una fórmula matemática, sino *asignación*. Esta asignación tiene el efecto de coger el valor de la variable `b1`, sumarle 1 y volver a almacenarla en la misma variable `b1`. El resultado final es que el valor de `b1` ha sido incrementado en 1.
También se pueden hacer cosas semejantes con variables de tipo `bool` y predicados:

```
bool b;
b = 2==3;
// b tendrá aquí el valor false
// puesto que 3 no es igual a 2
b = !b;
// ahora se cambia a false
```

Al contrario que en C y C++, donde una asignación es también una expresión, Xana no define la asignación como expresión. Por tanto el siguiente código, legal en C, no lo es en Xana:

```
a = b = 3 * 4;
```

cuyo efecto en `C` sería la asignación de 12 a la variable `b` y posteriormente la asignación del valor de `b` (ahora 12) a la variable `a`.

3.2.2.10 Precedencia

Cuando uno construye expresiones es importante tener en cuenta cómo van a ser evaluadas. Este concepto es el mismo concepto matemático de la asociatividad de operadores. Si las expresiones se evaluaran de izquierda a derecha el resultado final dependería del orden de los operandos y de los operadores: no sería lo mismo

`2 + 3 * 4 (20)` que `4 * 3 + 2 (14).`

Para evitar la ambigüedad se define la precedencia entre operadores, es decir, el orden de evaluación cuando son encontrados en una expresión. Las expresiones con operadores de mayor precedencia son evaluadas antes que los de precedencia más baja, mientras que operandos de igual precedencia son evaluados de izquierda a derecha. La precedencia de operadores de *Xana*, en orden descendente de precedencia, es:

- (cambio de signo)
<< >>
~
& | ^
* / %
+ -
< > >= <= <> !=
!

&&
||

En caso de querer especificar la precedencia de las operaciones manualmente, uno puede especificarlo mediante el uso de paréntesis. Una expresión rodeada de paréntesis siempre se evaluará antes que cualquier otra expresión no parentizada.

Siguiendo con el ejemplo de antes `2 + 3 * 4` se evalúa a `14`, pues la precedencia de `*` es mayor que la de `+`. Sin embargo `(2 + 3) * 4` es `20` pues los paréntesis alteran el orden de evaluación, forzando la evaluación de `2 + 3` primero.

3.2.2.11 Coerciones y desbordamientos

Como se ha visto, los operadores tienen cierta permisividad a la hora de aceptar operadores de tipos diferentes; por ejemplo, uno puede sumar un *byte* y un *dword*, obteniendo en este caso un *dword*. En realidad, para hacer esto posible (pues normalmente los microprocesadores y demás maquinas de ejecución de programas sí que son restrictivos en este aspecto) primero es necesario convertir el *byte* en un *dword* para poder sumarlo a otro *dword*.

Este tipo de conversiones (llamadas *coerciones*) son automáticas y las realiza el compilador sin intervención ni control por parte del usuario. Como automáticas que son, puede acontecer que los efectos no sean los deseados. Sea el siguiente código:

```
word a;
dword b;
b = 200000;
a = b;
```

En contra de lo esperado, el valor de a no será `200000`, pues `200000` no entra en el rango permitido de un *word*. En otras palabras, `200000` necesita de más de 16 bits para ser representado (concretamente es `110000110101000000`, con 18 bits); al ser una constante su tipo es *dword*, luego hasta aquí no hay problema. La conversión a *word* implica meter este valor en 16 bits, y el método empleado es quedarse con los dos `bytes` de menor peso. En este caso, con `0000110101000000`, prescindiendo de los dos bits de mayor peso (`11`). Esto implica que el valor de a será `3392`.

Cada vez que se requiera una coerción de un tipo a otro de menor tamaño, el compilador generará un aviso. Tales situaciones pueden ser fuente potencial de problemas y debieran al menos estar bajo control cuando no ser evitadas. El caso contrario, el paso de un tipo a otro de mayor tamaño (de `word` a `dword`, por ejemplo) no reviste peligrosidad alguna y siempre produce resultados previsibles: simplemente se extiende de signo, con lo que se pasa a tener el mismo valor, pero quizás con otra representación interna.

Un problema como el anterior puede aparecer en caso de que el resultado de una operación no entre en el rango de valores para su tipo. Por ejemplo, sea el siguiente código:

```
byte a;
```

```
byte b;
byte c;
c = 100;
b = 100;
a = b + c;
```

La suma de `b` y `c` produce lo que se denomina *desbordamiento*: el valor teórico `200` no entra en el rango de un *`byte`*, que es el tipo de la expresión. En tal caso, el valor se trunca (al igual que si fuera una coerción a *`byte`*); en este caso particular el resultado sería `-56`, que es el valor decimal de `11001000` en complemento a 2 sobre un *`byte`*.

3.2.3 Instrucciones

Hasta ahora, hemos cubierto la funcionalidad de *Xana* que le permite habilidades semejantes a las de una calculadora potente: evaluar expresiones y guardar resultados. Sin más funcionalidad la potencia de computación se ve limitada enormemente. Por ello *Xana* (y todo lenguaje de programación serio) posee el concepto de instrucción. Una instrucción es, conceptualmente, una orden que se pretende ejecute el computador. Por ejemplo, una asignación es una instrucción: se le dice con ella al computador que evalúe una expresión (a la derecha del '`=`') y asigne el valor resultante a la variable (a la izquierda de '`=`').

Al igual que ocurre con las expresiones, también se pueden construir instrucciones complejas a partir de instrucciones simples. De momento una instrucción simple es una asignación; más adelante veremos que hay más formas de instrucción simple[6].

3.2.3.1 Secuencia y bloque

La construcción más simple de instrucciones es la estructura secuencial. Consiste simplemente en colocar instrucciones una detrás de otra; el efecto será la ejecución de una tras la otra:

```
a = 3;          // primera instrucción
a = a * a + 1; // segunda instrucción
               // aquí a vale 10
```

La secuencia por sí misma no constituye una instrucción; esto es, no pueden colocarse una serie de instrucciones donde se pueda usar una sola instrucción). Para ello es necesario colocar la secuencia entre llaves:

```
{
  a = 3;          // primera instrucción
  a = a * a + 1; // segunda instrucción
                 // aquí a vale 10
}
```

[6] En realidad una expresión con un punto y coma detrás es una instrucción simple, al igual que lo es en C.

El total se denomina *bloque* y constituye una instrucción en sí mismo: donde se pueda usar una instrucción puede usarse un bloque de instrucciones. Utilizando este concepto uno puede entonces escribir:

```
{
  a = 3;            // primera instrucción
  {                 // segunda instrucción, que es un bloque
    a = a * a;
    a = a + 1;
   }                // aquí a vale 10
}
```

Una interesante propiedad de los bloques es la capacidad de poder declarar variables dentro de un bloque, variables sólo utilizables dentro del bloque:

```
byte b1;
{
  byte b2;
  b2 = 3;
}             // aquí b2 no esta definida
```

Dentro de un bloque uno puede incluso redefinir variables:

```
byte b;
b = 3;
{
  byte b;
  b = 4;
}             // aquí b valdrá 3
```

De esta manera uno puede aislar la creación de variables en el ámbito del código que la utiliza, sin que el resto tenga ni siquiera conocimiento de ello. Esta capacidad de aislamiento o ruptura en bloques es una de las bases del paradigma de programación estructurada[7].

3.2.3.2 Condicionales

Los constructores condicionales permiten decidir entre la ejecución o no ejecución de una porción de código según el valor de una expresión. Esto permite que el mismo código produzca ejecuciones diferentes dependiendo del valor de una o más variables.

1) **if-else**

La primera forma de instrucción compleja es la llamada estructura condicional o *if-else*, y tiene dos formas posibles. La primera de ellas, más simple, es:

```
if ( <predicado> ) <instrucción>
```

Si el predicado *<predicado>* (que recordemos era una expresión cuyo tipo es *bool* y por tanto evaluable a *true* o *false*) se evalúa a *true* la instrucción

[7] Metodología de programación que data de los años 70, hoy superada por la programación orientada a objetos pero todavía válida para proyectos pequeños.

<instrucción> se ejecuta. Y sólo en ese caso: si se evalúa a *false <instrucción>* no será ejecutada:

```
bool cond;
cond = a > b;
if (cond) a = b;
```

O bien

```
if (a > b) a = b
```

O bien

```
if (a > b)
{
  a = b;
}
```

La segunda variante, ligeramente más avanzada, es:

```
if ( <predicado> ) <instrucción-a> else <instrucción-b>
```

En este caso, una evolución del anterior, <instrucción-a> se ejecuta sí y sólo sí <predicado> se evalúa a true. Y si (y sólo sí) se evalúa a false <instrucción-b> será ejecutada. Por tanto la ejecución de ambas instrucciones es mutuamente excluyente y siempre se ejecuta alguna:

```
if (a > b)
{
  a = b;
}
else
{
  a = 0;
}
// si a era mayor que b ahora vale igual que b
// en caso contrario a vale 0
```

2) **switch**

Generalizando, una construcción *if-else* permite asociar una instrucción a cada uno de los posibles valores de una expresión; al ser una expresión booleana los posibles valores son dos (*true* y *false*) y por tanto disponemos de dos posibles instrucciones a ejecutar.

Si extendemos este concepto al ámbito de una expresión de tipo *dword* en lugar de una de tipo *bool* (predicado), y añadimos cierta flexibilidad a la hora de establecer la correspondencia entre valores e instrucciones tenemos la filosofía del constructor *switch*[8]. La apariencia general es:

```
switch ( <expresion-switch> )
{
  case <expresion-1> : <instrucciones-1>
  case <expresion-2> : <instrucciones-2>
  .
  .
```

[8] Denominado *case* en otros lenguajes, como por ejemplo *pascal*

```
  .
  case <expresion-n> : <instrucciones-n>
  default : <instrucciones-default>
}
```

El funcionamiento, al igual que en *C*, no es tan intuitivo como pudiera parecer en un primer momento: primeramente se evalúa *<expresion-switch>*. Después, se va comparando con la evaluación de cada expresión de *case* en orden (*<expresion-1>, <expresion-2>,* ...) hasta encontrar una que evalúe al mismo valor. En tal caso la ejecución continua en la primera instrucción del *<instrucciones-x>* asociada al *case*. Si ninguna expresión evalúa al valor de la expresión del *switch*, la ejecución continúa en la primera instrucción de *<instrucciones-default>*[9].

La parte extraña es que una vez ejecutadas las instrucciones asociadas al `case` *la ejecución continúa en la primera instrucción del case siguiente*. Es decir:

```
dword d;
switch (s)
{
  case 1: d = 2;
  case 2: d = 4;
  default: d = 100;
}
```

En el código anterior para un valor de `s = 2`, primeramente se asigna `4` a `d`, y después se le asigna `100`. Este comportamiento es una herencia de *C*, y por regla general trae más problemas que ventajas. En todo caso, para solucionar esto, tanto *C* como *Xana* proveen una instrucción especial: *break*.

break es una instrucción que provoca un salto en la ejecución; *break* sólo se puede utilizar en el ámbito de una sentencia *switch* o en el de un bucle (ver más adelante). En tal contexto el funcionamiento de *break* es simple; provoca que la ejecución continúe en la siguiente instrucción fuera del ámbito:

```
dword d;
switch (s)
{
  case 1:
    d = 2;
    break;
  case 2: d = 4;
    break;
  default:
    d = 100;
    break;
}
```

En este caso, con un valor de `s = 1`, se asigna `2` a `d` y seguidamente la instrucción *break* provoca que la ejecución prosiga en la instrucción inmediatamente siguiente

[9] A diferencia de *C*, *Xana* obliga a utilizar un default. En *C* es opcional.

a la llave cerrada (*}*) que cierra el *switch*. El *break* asociado al *default* no cambia la manera en que el *switch* es ejecutado; se suele poner sólo por coherencia.

3.2.3.3 Bucles

Otro tipo de constructor de instrucciones son los bucles. Permiten ejecutar un conjunto de instrucciones cierto número de veces, controlando el número mediante un predicado. *Xana* soporta los tres tipos de bucle definidos en *C*[10], con sintaxis y semántica casi idéntica: *while*, *do-while* y *for*.

1) **while**

El más sencillo de los bucles es el bucle while, que presenta la siguiente apariencia:

```
while ( <predicado> ) <instrucción>
```

La semántica es sencilla en extremo:

1. Se evalúa *<predicado>*
2. Si el resultado es *true* se ejecuta *<instrucción>*. Si es *false*, la ejecución sigue en la instrucción posterior a *<instrucción>*
3. Se vuelve al punto 1

Sirva como ejemplo un sencillo código para calcular el factorial de un número:

```
dword d;
dword i;
dword fac;
i = 1;
fac = 1;
while (i <= d)
{
  fac = fac * i;
  i = i + 1;
}
// fac contiene aquí el factorial de d
```

2) **do - while**

Una variante del anterior, apenas utilizada, es el bucle do-while. Su sintaxis es:

```
do <instrucción> while ( <predicado> )
```

Su semántica es semejante a la del bucle while:

1. Se ejecuta *<instrucción>*
2. Se evalúa *<predicado>*. Si es *true* se continúa en el punto 1. Si se evalúa a *false*, la ejecución continúa en la instrucción posterior al *while*.

[10] Al contrario que *C*, *Xana* no soporta el uso de *goto*. Tal instrucción se considera maldita y por tanto su ausencia es premeditada.

Como se puede apreciar, la diferencia está en el orden de evaluación del predicado. En un bucle *do-while <instrucción>* se ejecuta al menos una vez; en un bucle *while <instrucción>* puede no ejecutarse.

3) **for**

Como se puede ver, en los bucles *while* y *do-while* se especifica una condición de fin de bucle (el predicado), pero en ningún sitio se actualiza el estado del predicado. Se supone que todo bucle tiene que terminar (a veces no), y por tanto la ejecución de las instrucciones dentro del bucle tendrán algún efecto sobre el predicado, de tal manera que eventualmente se evaluará a *false*, dando por terminado el bucle. Se tendrán por tanto una serie de instrucciones que actúan sobre el predicado, y una serie de instrucciones que forman el bucle en sí. En el ejemplo de factorial `fac = fac * i` forma el cuerpo del bucle, el código necesario para calcular el factorial. `i = i + 1` tiene el efecto de cambiar el valor de evaluación del predicado haciendo que este eventualmente se evalúe a *false*.

Además, es preciso cierto código adicional para dar valor inicial a las variables que forman parte del bucle, tanto las que controlan el valor del predicado como las que sirven para los cálculos. En el ejemplo que nos ocupa, serían las asignaciones `i = 1` y `fac = 1`.

El constructor de bucle `for` separa las inicializaciones y las actualizaciones del predicado del cuerpo del bucle, añadiendo claridad al código. Su sintaxis es:

```
for ( <inicializaciones> ;
      <predicado> ;
      <instrucciones-actualizacion> )
<instrucción-bucle>
```

Donde tanto *<inicializaciones>* como *<instrucciones-actualizacion>* son una serie de instrucciones separadas por comas, pudiendo no haber ninguna. La manera en que un bucle `for` es tratado es la siguiente:

1. Se ejecutan las instrucciones de *<inicializaciones>*.
2. Se evalúa *<predicado>*. Si es false la ejecución continúa en la instrucción posterior al *for*. En caso contrario se ejecuta *<instrucción-bucle>*
3. Se ejecutan las instrucciones de *<instrucciones-actualizacion>*
4. Se continúa en el punto 2

El ejemplo de factorial, hecho con un bucle `for`, quedaría:

```
dword d;
dword i;
dword fac;
for (i = 1, fac = 1; i <= d; i = i + 1)
{
 fac = fac * i;
}
// fac contiene aquí el factorial de d
```

O, más simplificado:

```
dword d;
dword i;
dword fac;
for (i = 1, fac = 1;
     i <= d;
     i = i + 1, fac = fac * i;)
// fac contiene aquí el factorial de d
```

Como puede verse, la versatilidad[11] del bucle *for* le confiere gran potencia. Es incluso posible meter un programa en las cláusulas de un bucle *for*.

3.2.4 Funciones

Como ya se comentó anteriormente, lenguajes como C (en el cual se basa Xana) tenían en su fase de diseño el paradigma de programación descendente o estructurada como inspiración. Esto implica una facilidad inherente para subdividir un problema en subproblemas, los cuales son solucionados por separado y de manera independiente, y sus subsoluciones son combinadas para dar solución al problema original.

Tanto C como Xana proveen de dos mecanismos para conseguir tal subdivisión: el uso de bloques de instrucciones (ya comentado) y el uso de funciones. Desde le punto de vista de Xana una función es una entidad a la cual se le suministran unos valores y devuelve otro valor. Esto concuerda con el concepto matemático de función, de tal modo que un problema puede ser subdividido en subproblemas, cada uno de los cuales puede ser resuelto por una función.

Otro aspecto interesante de la subdivisión en funciones es la reutilización de código: una misma función puede resolver infinitas instancias de problema, sólo variando el valor de los parámetros de entrada.

La sintaxis de una definición de función es:

```
<tipo-retorno> <nombre> ( <parametros> ) { <instrucciones> }
```

Donde:

- *<tipo-retorno>* es el tipo de la función, o del valor que retorna la función
- *<nombre>* es el nombre simbólico de la función, que será usado para utilizar la misma
- *<parametros>* es una serie de declaraciones de variables separadas por comas, que representan los parámetros de la función. La serie puede ser vacía, significando que la función no toma parámetros. Dentro de la función los parámetros pueden ser invocados como otra variable cualquiera.

[11] Nótese que la palabra versátil no está reconocida en este momento por la Real Academia de la Lengua con la acepción utilizada en el documento (la Real Academia sólo reconoce actualmente la acepción para describir a una persona variable, lo que vulgarmente se denomina veleta), pero adelantándonos a su inminente modificación, ya la utilizamos con su nueva acepción.

- *<instrucciones>* es una secuencia de instrucciones que representa el cuerpo de la función. También se admiten declaraciones de variables.

Entre las instrucciones del cuerpo se puede usar la instrucción *return*, la cual toma una expresión. Tiene como efecto la asignación de la evaluación de tal expresión como valor de retorno de la función, seguido del fin de la ejecución de la misma, retornando el control al punto donde la función fue invocada.

Se da el hecho de que toda instrucción ha de estar englobada en una función, de tal manera que dentro de un fichero de código fuente sólo se admiten definiciones de variables y funciones[12].

A manera de ejemplo podemos construir fácilmente una función que calcule el factorial de un número dado:

```
dword factorial (dword n)
{
  dword fac;
  dword i;
  for (i = 1, fac = 1; i <= n; i = i + 1)
  {
    fac = fac * i;
  }
  return fac;
}
```

Este bloque de código puede utilizarse cuantas veces sea menester en cualquier otra parte del código.

3.2.4.1 Invocación, retorno y efectos laterales

Una vez definida una función podemos utilizarla como si fuera un constructor de expresiones en cualquier otra parte del código. De hecho una invocación a función es legal donde una expresión lo sea:

```
dword a;
a = 234;
dword res;
res = factorial (a);
```

En el ejemplo anterior se ve un ejemplo de invocación de una función[13], donde se utiliza para calcular el factorial de 234. Es evidente que la misma función puede ser usada para calcular el factorial de cualquier otro número. Factorial (a) es la invocación a la función: se especifica el nombre, y entre paréntesis se especifican los parámetros a usar en la invocación: serán necesarias tantas expresiones como parámetros se hayan especificado en la definición de la función, asignándose los valores de las expresiones a los parámetros especificados antes de comenzar la ejecución en sí (en este caso, un solo parámetro).

[12] Al igual que ocurre con *C*. Pero a diferencia de este, en *Xana* no hay una función de entrada, pues por su especial idiosincrasia no existe un punto de entrada único (en *C* hay un punto de entrada único y necesario, la función *main()*)
[13] A diferencia de *C*, *Xana* permite la invocación de funciones definidas más adelante en el fichero fuente

Cabe la posibilidad de que la función no retorne valor alguno; en tal caso el tipo de la misma se especifica como *void*. En todo caso, dentro del cuerpo de la función, es posible especificar tantos *return* como uno considere necesario, si bien en caso de que la función no sea *void* ésta ha de terminar obligatoriamente en un *return*. Si la función es *void*, la sintaxis de return es simplemente *return*; sin especificar expresión:

```
void doNothing (dword i)
{
  switch (i)
  {
    case 1:
      return;
    case 2:
    default:
      return;
  }
}
```

Una función *void* no puede ser usada como parte izquierda de una asignación, pues no es una expresión; No obstante cualquier función, y no sólo las de tipo *void*, pueden ser usadas como instrucciones, sin ser utilizadas como expresiones:

```
byte func1 ()
{
  ...
}
void func2 ()
{
  ...
}
...
void f ()
{
  byte b;
  b = func1 (); // esto es legal
  func2 ();  // esto también
  func1 (); esto también lo es
}
```

Evidentemente una función que no retorne nada es inútil, pues una función aparentemente sólo vale para calcular un valor en base a los valores asignados a sus parámetros y retornarlo; si la función no retorna nada, ¿para qué vale?

La respuesta está en los posibles efectos laterales de una función: el código de una función puede hacer referencia a las variables declaradas fuera de toda función, y por tanto modificar su valor.

```
dword dw;
void func ()
{
 dw = 34;
}
void act ()
```

```
{
  dw = 0;
  func ();
  // aquí dw tiene el valor 34
}
```

3.2.4.1.1 Recursividad

En el cuerpo de una función se puede poner cualquier instrucción válida, que puede incluir llamadas a funciones. De hecho, el cuerpo de una función puede contener llamadas a sí misma. Esta capacidad[14] permite programar algoritmos que de otra manera (en base a bucles y otros constructores) sería más engorroso escribir, cuando no imposible. Sirva como ejemplo esta función para el cálculo del n-esimo valor de la serie de Fibonacci (donde cada elemento es la suma de los dos anteriores, siendo el primero y el segundo ambos 1):

```
dword fib (dword n)
{
  if (n < 2)
  return 1;

  return fib (n - 2) + fib (n - 1);
}
```

Obsérvese la simplicidad del código. Tal simplicidad es sólo aparente, pues los algoritmos recursivos suelen ser voraces consumidores de recursos. En el caso de esta función el tiempo empleado para el cálculo crece exponencialmente con el valor de n. También existe el peligro de encontrarse con una recursión infinita: la función siempre acaba invocándose a sí misma. La ejecución de tal función provocará un error de ejecución de la máquina virtual por desbordamiento de pila.
No hay que entender lo anterior como un veto a los algoritmos recursivos: suelen ser mucho más claros de programar que un equivalente con bucles, y a veces son la única solución.

3.2.4.2 Paso por valor, paso por referencia

Como ya se ha dicho, los parámetros de una función pueden considerarse como variables, accesibles sólo desde dentro de la función. Tales parámetros toman sus valores iniciales de la lista de expresiones especificada en cada invocación de la función. Sea la función `f`:

```
void f (dword d)
{
  d = 0;
}
```

Tal función cambia el valor del parámetro `d`, pero tal cambio sólo afecta al cuerpo de la función:

```
dword z;
z = 10;
```

[14] no todos los lenguajes la poseen, aunque los que no la poseen son bastante antiguos: como el COBOL

```
f (z);
// z sigue valiendo 10
```

Esta manera de paso de parámetros de denomina paso por valor, pues lo único que se suministra a la función son los valores de los parámetros.

Xana admite otro modo de paso de parámetros, en el cual se pasa una variable como tal, no sólo su valor. Para conseguir tal efecto se modifica ligeramente la definición de la función:

```
void f (dword& d)
{
  d = 0;
}
```

El ampersand (`&`) que sigue al tipo del parámetro especifica que lo que se pasa a la función no es un valor: es una referencia a la variable[15]. Esto tiene el efecto de poder cambiar el valor de tal variable desde dentro de la función:

```
dword z;
z = 10;
f (z);
// z valdrá 0
```

Por lo demás el cuerpo de la función no varía en absoluto, como se puede observar. Mediante el uso de paso de parámetros por referencia es posible programar funciones sin apenas efectos laterales, o programar funciones con más de un valor de retorno.

En el caso de un parámetro pasado por valor es posible utilizar una expresión como parámetro en la invocación:

```
void f (dword d)
{
  d = 3;
}
void ff (dword& d)
{
  d = 3;
}
...

void func ()
{
  dword d;
  f (d);  // permitido
  ff (d); // permitido
  f (d + 1); // permitido
  ff (d + 1); // no permitido
}
```

[15] La sintaxis está tomada de C++

3.2.4.3 Variables locales vs. Variables globales: visibilidad

Como ya se ha explicado, se pueden definir variables fuera de toda función, dentro del cuerpo de una función o dentro de un bloque de código. A las variables definidas fuera de toda función se les denomina variables globales; al resto, variables locales. Toda variable definida tiene un ámbito de definición:

- *Variables globales*: desde el punto de su definición hasta el fin del fichero fuente
- *Variables locales*: desde el punto de su definición hasta el fin del bloque donde fueron definidas. Cualquier bloque que se encuentre entre medias cae dentro del ámbito. A efectos de ámbitos, una función define un bloque correspondiente a su cuerpo.

```
dword d1;  // inicio de ámbito de d1
dword d2;  // inicio de ámbito de d2
void f (byte b)
{
  // inicio de ámbito de b
  byte b2; // inicio de ámbito de b2
  {
    byte b3;  // inicio de ámbito de b3
    ...
  } // fin de ámbito de b3
...
} // fin de ámbito de b y b2
...
// fin de ámbito de d1 y d2
```

Como ya se explicó, una variable definida dentro de un ámbito con un nombre de otra variable definida en un ámbito exterior enmascara (o redefine) a ésta última:

```
dword d1;
// d1 se refiere aquí al dword
void f ()
{
  ...
  // hasta aquí d1 se refiere al dword
  byte d1;
  // a partir de aquí d1 se refiere a la variable global de tipo
byte
  ...
}
// aquí d1 se refiere de nuevo al dword
```

Las variables definidas no pueden ser referenciadas desde cualquier sitio: sólo pueden ser referenciadas dentro de su ámbito. En otro caso el compilador generará un error de 'variable no definida'.

Dado que es posible especificar el ámbito de las variables, es una buena práctica de programación el restringir aquel al máximo, esto es, definir las variables solamente donde se necesiten. Esto ayuda en gran medida a mantener el código en unidades funcionales fácilmente reutilizables, puesto que disminuye la interdependencia entre partes del código.

En todo caso, las variables globales debieran definirse sólo cuando sean necesarias. En el caso de *Xana* sólo las variables globales pueden llevar modificadores de almacenamiento, luego sólo estas pueden ser ligadas al bus de campo, o dotadas de persistencia.

3.2.5 Objetos nativos

Xana es un lenguaje orientado al control en tiempo real; por tanto, la ejecución del código ha de ser lo más rápida posible. Hay además ciertas partes del código del bucle de control que serán, en cierta manera, comunes a cualquier algoritmo de control. Estas partes comunes, como cualquier interfaz con el mundo real (modelos de secuenciador, por ejemplo) o cualquier otra librería general (como una librería de funciones matemáticas complejas) son susceptibles de ser aceleradas al máximo sin perder flexibilidad, pues en cierto modo una vez escritas y probadas su código no cambiará.

Por tal razón, toda esa funcionalidad podría ser movida a la capa de la máquina virtual como parte de ella, y por tanto con categoría de inmutable. Tal enfoque es quizás demasiado radical:

- El tamaño de la máquina virtual crecería con la demanda de nueva funcionalidad. La VM tendría que incluir toda la funcionalidad necesaria para cada programa.
- Cada vez que se requiera aumentar la funcionalidad, será necesario recompilar y reinstalar la VM.

Es evidente que tal enfoque no aporta comodidad alguna. Por ello se desarrolló una arquitectura de componentes de quita y pon, que sólo son cargados (y permanecen cargados) en memoria cuando son usados. La VM puede cargar cualquier número de ellos, y es el código escrito en *Xana* el que utiliza directamente estos componentes.

Xana adopta un paradigma basado en objetos para utilizar estos componentes: cuando se declara una variable de un tipo no estándar el compilador comprueba la existencia de un componente con ese nombre (de no existir el compilador generará un error); a partir de tal momento la variable será considerada una instancia de tal componente. Un componente puede verse como un conjunto de variables ocultas y una serie de funciones que actúan sobre estos. En otras palabras, una variable definida de tal manera es un objeto, según la terminología al uso en lenguajes como *C++* o *java*. De hecho, el uso que se le puede dar a tal variable es radicalmente distinto al resto: sólo se puede usar para invocar métodos sobre ella, mediante el operador '->':

```
Machine s1; // declara un secuenciador
Machine s2; // declara otro secuenciador, distinto e independiente
del anterior
...
void f ()
{
  ...
  s1->init (); // inicializa el secuenciador 1
```

```
  s2->init (); // inicializa el secuenciador 2
  ...
}
```

La funcionalidad de un componente (las funciones que pueden ser invocadas) vienen definidas por el propio componente, de tal manera que un componente es una entidad auto contenida. Ante una invocación, el compilador interroga al componente para averiguar si existe tal método, y si los parámetros pasados son correctos en número y tipo.

3.2.5.1 Paso y Retorno de Componentes a Funciones

Los Componentes con parámetros validos de entrada a las funciones, pudiendo combinarse con cualquier otro tipo de parámetros. Esto permite más flexibilidad a la hora de realizar funciones.

Podría definirse por ejemplo una función que vaya a posición independientemente del eje del que se trate:

```
void Posiciona( dword PosicionDestino, Ipos_Sew Variador, bool&
Run )
{
  Run = true;
  Variador->GoToPosition( PosicionDestino );
}
```

De la misma manera se pueden definir funciones que devuelvan componentes, como por ejemplo:

```
Machine MaquinaPorNumero( dword Numero )
{
  switch( Numero )
  {
    case 1:
      return TFC_01;
    case 2:
       return TFC_02;
    default:
       return TFC_03;
  }
}
```

3.2.5.1.1 Listado de componentes del Sistema de Control Galileo

LIBRERÍA DE COMPONENTES DE ACCESO AL SISTEMA (Sysobj.dll)	
1	OBJETO SYSTEM
2	OBJETO MACHINE
3	OBJETO CONFIG
4	OBJETO POSITIONMAP
5	OBJETO DIMENSIONALMAP
6	OBJETO DIMENSIONALMAPEX
LIBRERÍA DE COMPONENTES COMUNES DE ACCESO AL HARDWARE (ComHard.dll)	

7	OBJETO OP PILZ
8	OBJETO OP UNIOP
9	OBJETO ENCODER ENCOM
10	OBJETO ENCODER T+R
11	OBJETO ENCODER PROFIBUS CLASE 2
12	OBJETO ENCODER CANOPEN CLASE 2
13	OBJETO VARIADOR IPOS SEW
14	OBJETO VARIADOR IPOS SEW EX
15	OBJETO VARIADOR MOVITRAC
16	OBJETO VARIADOR LENZE 9300
17	OBJETO VARIADOR LENZE PLC 9300
18	OBJETO VARIADOR LENZE ECS
19	OBJETO VARIADOR LENZE L-FORCE
20	OBJETO VARIADOR LENZE SMV VECTOR
21	OBJETO VARIADOR LENZE 8400
22	OBJETO ESCÁNER ISCANNER
23	OBJETO ESCÁNER SICK PUERTO SERIE
24	OBJETO ESCÁNER SICK TCP
25	OBJETO ESCÁNER SICK PCP
26	OBJETO ESCÁNER SICK PB
27	OBJETO ESCÁNER LEUZE 504i
28	OBJETO ESCÁNER LEUZE BCL34
29	OBJETO BÁSCULA DE PESAJE MICROGRAM
30	OBJETO ANTENA TAG KL6001 RFM12
31	OBJETO ANTENA TAG KL6001 UDP1
32	OBJETO INDITEX SE
33	OBJETO ESCÁNER VOLUMÉTRICO METTLER TOLEDO
34	OBJETO ELECTROVÍA LJU RAILBUS SYSTEM
LIBRERÍA DE COMPONENTES DE COMUNICACIONES (Comcomponentes.dll)	
35	OBJETO ORACLE
36	OBJETO SERVERSOCKET
37	OBJETO SERVERSOCKETRAW

3.3 Tabla de palabras reservadas

```
bool
break
byte
case
default
do
double
dword
else
false
for
if
```

input
long
namespace[16]
output
persistent
return
shared
switch
true
void
while
word

[16] Palabra reservada para uso futuro

4 COMPONENTES DEL SISTEMA DE CONTROL GALILEO

4.1 Librería de componentes de acceso al sistema (SYSOBJ.dll)

Aunque la exposición de los diferentes objetos que se encuentran en esta biblioteca puede resultar un poco árida en este momento en que el lector no se ha centrado todavía en los conceptos de programación, resulta imprescindible como guía de referencia y para poder entender el resto de apartados.

4.1.1 CONFIG

4.1.1.1 Interface

Este componente permite utilizar un archivo para almacenar y extraer parámetros de configuración. Este archivo debe tener una estructura de las siguientes características:

- El archivo está compuesto de líneas numeradas.
- Cada línea tiene el formato `Parametro = Valor    // Comentario`
- `Parámetro` y `Valor` son valores numéricos enteros, el comentario debe residir en la misma línea.

Como ejemplo de archivo de configuración, se puede mostrar el correspondiente a un transelevador:

```
1 = 2500000 // Valor máximo en pulsos para el eje X
2 = 2200000 // Valor en pulsos cambio velocidad adelante X
3 =  500000 // Valor en pulsos cambio velocidad atras X
4 =  200000 // Valor mínimo en pulsos para el eje X
5 =     256 // Intervalo en pulso Set Centraje X
6 =0        // Intervalo en pulsos Reset Centraje X
7 = 2100    // Intervalo en pulsos ventana posicionamiento X
8 =0        // Valor incremento máximo en pulsos para un ciclo X
9 =0        // Valor incremento mínimo en pulsos para un ciclo Y
```

Debe hacerse notar que para trabajar con estos archivos existen dos métodos vitales. El método Load es el primero que se debe invocar y debe de hacerse siempre. Este método carga el archivo de disco, y a partir de este momento todas las consultas o modificaciones se realizarán sobre memoria. Si se ha modificado el archivo se debe invocar `Sync` para archivarlo y no perderlas si la aplicación finaliza.
Para manejar este tipo de archivos en los cuales se colocarán los valores que se deseen configurar (valores que el programa no debería recibir por código) este componente expone los siguientes métodos.

El interface de este componente es el siguiente:

Class	**Config**
BUS IN	0 bytes
BUS OUT	0 bytes

Listado métodos:

1	`bool Add(dword Parameter, dword Value)`
2	`void Del(dword ParameterNumber)`
3	`bool Exists(dword ParameterNumber)`
4	`bool Load(string File)`
5	`bool LoadAndTrack(string File)`
6	`bool Modify(dword ParameterNumber, dword Value)`
7	`bool Sync()`
8	`dword Value(dword ParameterNumber)`

Descripción de métodos:

1

```
bool Add(dword ParameterNumber,
         dword Value)
```

Descripción general

Este método añade o sobrescribe un parámetro en memoria.

Retorno

`true`
Si la operación se ha realizado con éxito.
`false`
Si no se ha cargado ningún archivo previamente a la invocación.

Parámetros

1. `dword ParameterNumber`
   Número de parámetro que se desea consultar.
2. `dword Value`
   Valor que se desea asignar al parámetro.

2

```
void Del(dword ParameterNumber)
```

Descripción general

Este método elimina un parámetro determinado del fichero.

Retorno

`void`
Este método no retorna nada.

Parámetros

1. `dword ParameterNumber`
   Número de parámetro que se desea consultar.

3

```
bool Exists(dword ParameterNumber)
```

Descripción general

Este método consulta la existencia de un parámetro en el archivo de configuración.

Retorno

`true`
Si el parámetro existe en el archivo de configuración.
`false`
Si el parámetro NO existe en el archivo de configuración.

Parámetros

1. `dword` `ParameterNumber`
   Número de parámetro que se desea consultar.

4

```
bool Load(string File)
```

Descripción general

Carga en memoria el archivo especificado. Este método debe ser invocado en las rutinas de arranque ya que en el código cíclico supondría una carga de tiempo de ciclo el acceso al sistema de archivos.

Retorno

`true`
Si la carga del fichero ha tenido éxito.
`false`
Si la carga del fichero no se ha podido finalizar correctamente.

Parámetros

1. `string` `File`
   Nombre del archivo que se desea cargar (con ruta).

5

```
bool LoadAndTrack(string File)
```

Descripción general

Carga en memoria el archivo especificado. Este método debe ser invocado en las rutinas de arranque ya que en el código cíclico supondría una carga de tiempo de ciclo el acceso al sistema de archivos. La diferencia de este método con el otro método de carga estriba en que una vez cargado, el componente permanece pendiente de cualquier modificación que sufra el archivo de carga, y si se produce alguna, procede a recargar los datos del mismo. Esto evita tener que parar y rearrancar Galileo para recargar una configuración.

Retorno

`true`
Si la operación se ha realizado con éxito.
`false`
La operación NO se ha realizado con éxito, es decir, el archivo no ha sido cargado.

Parámetros

`String` `File`
Nombre del archivo que se desea cargar (especificando la ruta).

6

```
bool Modify(dword ParameterNumber,
            dword Value)
```

Descripción general

Este método añade o sobrescribe un parámetro en memoria, o lo crea si no existiese.

Retorno

`true`
Si la operación se ha realizado con éxito.
`false`
Si no se ha cargado ningún archivo previamente a la invocación.

Parámetros

1. `dword ParameterNumber`
   Número de parámetro que se desea consultar.
2. `dword Value`
   Valor que se desea asignar al parámetro.

7

```
bool Sync()
```

Descripción general

Este método archiva en disco todos los cambios que se han producido en el archivo de configuración.

Retorno

`true`
Si la operación se ha realizado con éxito.
NOTA: esta operación se ejecuta **sincrónicamente**.
`false`
La operación NO se ha realizado con éxito, es decir, el archivo no ha sido cargado.

Parámetros

Este método no tiene parámetros de entrada.

8

```
dword Value(dword ParameterNumber)
```

Descripción general

Retorna el valor configurado para un parámetro determinado.

Retorno

`dword`
Retorna el valor configurado para el parámetro.

Parámetros

1. `dword ParameterNumber`
   Número del parámetro que se desea consultar.

4.1.2 DIMENSIONALMAP

4.1.2.1 Interface

Este componente proporciona un mecanismo para almacenar un mapa multidimensional donde el número de dimensiones varía desde 1 hasta un valor indeterminado. Gracias a este objeto, es posible almacenar posiciones bidimensionales y tridimensionales en memoria para su consulta o modificación y salvarlas / cargarlas a / desde un fichero. El formato de este fichero, pese a ser texto plano, no está diseñado para ser modificado vía un editor de textos convencional, por lo que se recomienda que no se intente editar para añadir / eliminar / modificar los valores de las posiciones que contiene.

Ha de hacerse notar que, a pesar de que este componente está diseñado para soportar mapas con n dimensiones, a la mente humana común se le hace muy difícil trabajar con más de 4 dimensiones, y que la mayor utilidad de este componente se basa en manejar información sobre sistemas de 2 y 3 dimensiones.

Este es el motivo por el cual se le ha provisto únicamente de métodos para trabajar con esas dos dimensiones desde *Xana*, dado que muy raramente será necesario que trabaje con un número de dimensiones mayor. No obstante, si se diese la necesidad de tal evento, el componente puede ser modificado en poco tiempo para permitir trabajar con cualquier dimensión.

Al igual que los mapas que lo preceden, este mapa también lleva incorporado unos *Offsets*, que en este caso es un vector cuyo tamaño es la dimensión del mapa.

Estos offsets sirven para ajustar el valor pedido, y siempre se suman a los valores almacenados en memoria, de forma transparente para el usuario que los consulta.

El interface de este componente es el siguiente:

Class	DimensionalMap
BUS IN	0 bytes
BUS OUT	0 bytes

Listado de métodos:

1	`bool Add(dword CoordinateX,`
	`dword CoordinateY,`
	`dword ValueX,`
	`dword ValueY)`

2	`bool Add3D(dword CoordinateX,`
	`dword CoordinateY,`
	`dword CoordinateZ,`
	`dword ValueX,`
	`dword ValueY,`
	`dword ValueZ)`

3	`void BeginUpdate()`
4	`bool Clear()`
5	`bool Del(dword CoordinateX,` `dword CoordinateY)`
6	`bool Del3D(dword CoordinateX,` `dword CoordinateY,`

	dword CoordinateZ)
7	void EndUpdate()
8	bool Exists(dword CoordinateX, dword CoordinateY)
9	bool Exists3D(dword CoordinateX, dword CoordinateY, dword CoordinateZ)
10	bool GetNearestPosition(dword PositionX, dword PositionY, dword &NearestX, dword &NearestY)
11	bool GetNearestPosition3D(dword PositionX, dword PositionY, dword PositionZ, dword &NearestX, dword &NearestY, dword &NearestZ)
12	bool GetPositionFollowingGradient(dword CurrentPositionX, dword CurrentPositionY, dword DeltaX, dword DeltaY, dword &PosX, dword &PosY)
13	bool GetPositionFollowingGradient3D(dword CurrentPositionX, dword CurrentPositionY, dword CurrentPositionZ, dword DeltaX, dword DeltaY, dword DeltaZ, dword &PosX, dword &PosY, dword &PosZ)
14	bool IOFinished()
15	bool Load(string File)
16	bool LoadAndTrack(string File)
17	bool Modify(dword CoordinateX, dword CoordinateY, dword ValueX, dword ValueY)
18	bool Modify3D(dword CoordinateX, dword CoordinateY, dword CoordinateZ, dword ValueX, dword ValueY, dword ValueZ)
19	bool Position(dword CoordinateX, dword CoordinateY, dword &PositionX, dword &PositionY)
20	bool Position3D(dword CoordinateX, dword CoordinateY, dword CoordinateZ, dword &PositionX, dword &PositionY, dword &PositionZ)

21	`bool SetDimension(dword Dimension)`
22	`bool SetSearchRange(dword MaximumX, dword MaximumY)`
23	`bool SetSearchRange3D(dword MaximumX, dword MaximumY, dword MaximumZ)`
24	`bool Sync(string File)`
25	`dword CountStoredPositions(void)`

Descripción de métodos:

1

```
bool Add(dword CoordinateX,
         dword CoordinateY,
         dword ValueX,
         dword ValueY)
```

Descripción general

Este método añade o sobrescribe una coordenada de un mapa bidimensional en memoria. **Este método trabaja con lógica negada.**

Retorno

`false`
Si la operación se ha realizado con éxito.
`true`
Si la operación no ha podido ser finalizada.

Parámetros

1. `dword CoordinateX`
Coordenada X de la posición cuyo valor se quiere añadir/modificar.
2. `dword CoordinateY`
Coordenada Y de la posición cuyo valor se quiere añadir/modificar.
3. `dword ValueX`
Valor que se desea asignar a la coordenada X indicada.
4. `dword ValueY`
Valor que se desea asignar a la coordenada Y indicada.

2

```
bool Add3D(dword CoordinateX,
           dword CoordinateY,
           dword CoordinateZ,
           dword ValueX,
           dword ValueY,
           dword ValueZ)
```

Descripción general

Este método añade o sobrescribe una coordenada de un mapa bidimensional en memoria. **Este método trabaja con lógica negada.**

Retorno

`false`
Si la operación se ha realizado con éxito.
`true`

Si la operación no ha podido ser finalizada.

Parámetros

1. `dword CoordinateX`
Coordenada X de la posición cuyo valor se quiere añadir/modificar.
2. `dword CoordinateY`
Coordenada Y de la posición cuyo valor se quiere añadir/modificar.
3. `dword CoordinateZ`
Coordenada Z de la posición cuyo valor se quiere añadir/modificar.
4. `dword ValueX`
Valor que se desea asignar a la coordenada X indicada.
5. `dword ValueY`
Valor que se desea asignar a la coordenada Y indicada.
6. `dword ValueZ`
Valor que se desea asignar a la coordenada Z indicada.

3

```
void BeginUpdate()
```

Descripción general

La modificación/inserción o borrado de valores en los mapas dimensionales puede resultar en un cierto tiempo en que el componente debe actualizar su caché de acceso. Cuando se realizan modificaciones masivas a muchas posiciones, la suma de estos tiempos puede resultar en retardos inaceptables para el uso del mismo. Este método sirve para desactivar temporalmente las actualizaciones en la caché del componente mientras se llevan a cabo operaciones de mantenimiento de este tipo (muchas modificaciones consecutivas), a fin de que la caché no se actualice, salvando así ese tiempo extra. Una vez que se finalicen las modificaciones, se debe invocar al método complementario de este, `EndUpdate` a fin de refrescar la caché de forma manual.
NOTA IMPORTANTE: Si no se realiza la llamada a `EndUpdate` tras el uso de este método, el componente no refrescará su caché interna, por lo que el acceso a sus datos vera degradada su velocidad de forma significativa, y pudiera no ofrecer datos actualizados.

Retorno

`void`
Este método no retorna nada.

Parámetros

Este método no requiere parámetros de entrada.

4

```
bool Clear()
```

Descripción general

Este método limpia del mapa en memoria todos los valores de coordenadas previamente introducidos o cargados en él. **Este método trabaja con lógica negada.**

Retorno

`false`
Si la operación se ha realizado con éxito.

`true`
Si la operación no ha podido ser finalizada (no se pudo limpiar el mapa de valores).

Parámetros

Este método no requiere parámetros de entrada.

5

```
bool Del(dword CoordinateX,
         dword CoordinateY)
```

Descripción general

Este método elimina una coordenada de un mapa tridimensional. **Este método trabaja con lógica negada.**

Retorno

`false`
Si la operación se ha realizado con éxito.
`true`
Si la operación no ha podido ser finalizada.

Parámetros

1. `dword` `CoordinateX`

Coordenada X de la posición que se desea eliminar.

2. `dword` `CoordinateY`

Coordenada Y de la posición que se desea eliminar.

6

```
bool Del3D(dword CoordinateX,
           dword CoordinateY,
           dword CoordinateZ)
```

Descripción general

Este método elimina una coordenada de un mapa tridimensional. **Este método trabaja con lógica negada.**

Retorno

`false`
Si la operación se ha realizado con éxito.
`true`
Si la operación no ha podido ser finalizada.

7

```
void EndUpdate()
```

Descripción general

Este método sirve para indicar al componente que ha terminado una serie de operaciones de mantenimiento y que puede volver a actualizar la caché interna. Deberá ser invocado obligatoriamente tras una invocación a BeginUpdate. Véase nota sobre el método `BeginUpdate`.

Retorno

`void`
Este método no retorna nada.

Parámetros

Este método no requiere parámetros de entrada.

8

```
bool Exists(dword CoordinateX,
            dword CoordinateY)
```

Descripción general

Este método consulta la existencia de una posición de dos coordenadas en el mapa.

Retorno

`true`
Si las coordenadas consultadas existen en el mapa.
`false`
Si las coordenadas consultadas NO existen en el mapa.

Parámetros

1. `dword CoordinateX`
Coordenada X de la posición que se desea consultar.
2. `dword CoordinateY`
Coordenada Y de la posición que se desea consultar.

9

```
bool Exists3D(dword CoordinateX,
              dword CoordinateY,
              dword CoordinateZ)
```

Descripción general

Este método consulta la existencia de una posición de dos coordenadas en el mapa.

Retorno

`true`
Si las coordenadas consultadas existen en el mapa.
`false`
Si las coordenadas consultadas NO existen en el mapa.

Parámetros

1. `dword CoordinateX`
Coordenada X de la posición que se desea consultar.
2. `dword CoordinateY`
Coordenada Y de la posición que se desea consultar.
3. `dword CoordinateZ`
Coordenada Z de la posición que se desea consultar.

10

```
bool GetNearestPosition(dword PositionX,
                        dword PositionY,
                        dword &NearestX,
                        dword &NearestY)
```

Descripción general

Este método sirve para averiguar las coordenadas de la posición almacenada en un mapa más cercana a una posición dada. El mapa se supone bidimensional. Nótese que no es necesario que las coordenadas pasadas existan actualmente en el mapa. **Este método trabaja con lógica negada.**

Retorno

`false`
Si la operación se ha realizado con éxito, y en los parámetros `NearestX` y `NearestY` se han almacenado las coordenadas obtenidas como más cercanas a las dadas.
`true`
No se han podido obtener las coordenadas pedidas (por ejemplo, el mapa está vacío).

Parámetros

1. `dword PositionX`
Coordenada X a evaluar.
2. `dword PositionY`
Coordenada Y a evaluar.
3. `dword NearestX`
Coordenada existente en el mapa más cercana a la coordenada X indicada en el parámetro `PositionX`.
4. `dword NearestY`
Coordenada existente en el mapa más cercana a la coordenada Y indicada en el parámetro `PositionY`.

11

```
bool GetNearestPosition3D(dword PositionX,
                          dword PositionY,
                          dword PositionZ,
                          dword &NearestX,
                          dword &NearestY,
                          dword &NearestZ)
```

Descripción general

Este método sirve para averiguar las coordenadas de la posición almacenada en un mapa más cercana a una posición dada. El mapa se supone tridimensional. Nótese que no es necesario que las coordenadas pasadas existan actualmente en el mapa. **Este método trabaja con lógica negada.**

Retorno

`false`
Si la operación se ha realizado con éxito, y en los parámetros `NearestX`, `NearestY` y `NearestZ` se han almacenado las coordenadas obtenidas como más cercanas a las dadas.
`true`
No se han podido obtener las coordenadas pedidas (por ejemplo, el mapa está vacío).

Parámetros

1. `dword PositionX`
Coordenada X a evaluar.
2. `dword PositionY`
Coordenada Y a evaluar.

3. `dword PositionZ`
Coordenada Z a evaluar.
4. `dword NearestX`
Coordenada existente en el mapa más cercana a la coordenada X indicada en el parámetro `PositionX`.
5. `dword NearestY`
Coordenada existente en el mapa más cercana a la coordenada Y indicada en el parámetro `PositionY`.
6. `dword NearestZ`
Coordenada existente en el mapa más cercana a la coordenada Z indicada en el parámetro `PositionZ`.

12

```
bool GetPositionFollowingGradient(dword CurrentPositionX,
                                  dword CurrentPositionY,
                                  dword DeltaX,
                                  dword DeltaY,
                                  dword &PosX,
                                  dword &PosY)
```

Descripción general

Dado un mapa bidimensional, mediante este método se permite obtener las coordenadas de la posición a partir de una dada, siguiendo un vector de gradiente también dado. **Este método trabaja con lógica negada.**

Retorno

`false`
Si la operación se ha realizado con éxito, y en los parámetros `PosX` y `PosY` se han almacenado las coordenadas obtenidas.
`true`
No se han podido obtener las coordenadas pedidas (no existen en el mapa).

Parámetros

1. `dword CurrentPositionX`
Coordenada X de la posición origen.
2. `dword CurrentPositionY`
Coordenada Y de la posición origen.
3. `dword DeltaX`
Valor del vector gradiente en el eje X.
4. `dword DeltaY`
Valor del vector gradiente en el eje Y.
5. `dword PosX`
Se devuelve la coordenada X encontrada a partir de ese gradiente, siguiendo la coordenada X indicada en el parámetro `CurrentPositionX`.
6. `dword PosY`
Se devuelve la coordenada Y encontrada a partir de ese gradiente, siguiendo la coordenada Y indicada en el parámetro `CurrentPositionY`.

13

```
bool GetPositionFollowingGradient3D(dword CurrentPositionX,
                                    dword CurrentPositionY,
                                    dword CurrentPositionZ,
                                    dword DeltaX,
```

```
                                    dword DeltaY,
                                    dword DeltaZ,
                                    dword &PosX,
                                    dword &PosY,
                                    dword &PosZ)
```

Descripción general

Dado un mapa tridimensional, mediante este método se permite obtener las coordenadas de la posición a partir de una dada, siguiendo un vector de gradiente también dado. **Este método trabaja con lógica negada.**

Retorno

`false`
Si la operación se ha realizado con éxito, y en los parámetros `PosX` y `PosY` se han almacenado las coordenadas obtenidas.
`true`
No se han podido obtener las coordenadas pedidas (no existen en el mapa).

Parámetros

1. `dword CurrentPositionX`
Coordenada X de la posición origen.
2. `dword CurrentPositionY`
Coordenada Y de la posición origen.
3. `dword CurrentPositionZ`
Coordenada Z de la posición origen.
4. `dword DeltaX`
Valor del vector gradiente del eje X.
5. `dword DeltaY`
Valor del vector gradiente del eje Y.
6. `dword DeltaZ`
Valor del vector gradiente del eje Z.
7. `dword PosX`
Se devuelve la coordenada X encontrada a partir de ese gradiente, siguiendo la coordenada X indicada en el parámetro `CurrentPositionX`.
8. `dword PosY`
Se devuelve la coordenada Y encontrada a partir de ese gradiente, siguiendo la coordenada Y indicada en el parámetro `CurrentPositionY`.
9. `dword PosZ`
Se devuelve la coordenada Z encontrada a partir de ese gradiente, siguiendo la coordenada Z indicada en el parámetro `CurrentPositionZ`.

14

```
bool IOFinished()
```

Descripción general

Indica si el componente está realizando una operación de entrada/salida en ese momento. **Este método trabaja con lógica negada.**

Retorno

`False`
El componente aún ya ha finalizado una operación de E/S, por lo que se puede inicia otra operación sin riesgo
`true`
El componente aún no ha finalizado una operación de E/S, por lo que si se inicia

otra se puede producir un error

Parámetros

Este método no requiere parámetros de entrada.

15

```
bool Load(string File)
```

Descripción general

Carga en memoria el archivo especificado. Este método puede ser invocado tanto en las rutinas de arranque como en el código cíclico ya que la carga del fichero se realiza de forma asíncrona, pero ha de tenerse en cuenta que durante el tiempo que dure la carga, las consultas sobre posiciones aun no cargadas de fichero devolverán valores inconsistentes o por defecto. **Este método trabaja con lógica negada.**

Retorno

`false`
Si la carga del fichero ha tenido éxito.
`true`
Si la carga del fichero no se ha podido finalizar correctamente.

Parámetros

1. `string File`
   Nombre del archivo que se desea cargar (con ruta).

16

```
bool LoadAndTrack(string File)
```

Descripción general

Carga en memoria el archivo especificado. Este método debe ser invocado en las rutinas de arranque ya que en el código cíclico supondría una carga de tiempo de ciclo el acceso al sistema de archivos. La diferencia de este método con el otro método de carga estriba en que una vez cargado, el componente permanece pendiente de cualquier modificación que sufra el archivo de carga, y si se produce alguna, procede a recargar los datos del mismo. Esto evita tener que parar y rearrancar Galileo para recargar una configuración. **Este método trabaja con lógica negada.**

Retorno

`false`
Si la operación se realizó con éxito.
`true`
Si no se ha podido cargar el archivo.

Parámetros

1. `string File`
Nombre del archivo que se desea cargar.

17

```
bool Modify(dword CoordinateX,
            dword CoordinateY,
            dword ValueX,
            dword ValueY)
```

Descripción general

Este método sobrescribe una coordenada de un mapa bidimensional en memoria. **Este método trabaja con lógica negada.**

Retorno

`false`
Si la operación se ha realizado con éxito.
`true`
Si la operación no ha podido ser finalizada.

Parámetros

1. `dword CoordinateX`
Coordenada X de la posición cuyo valor se quiere modificar.
2. `dword CoordinateY`
Coordenada Y de la posición cuyo valor se quiere modificar.
3. `dword ValueX`
Valor que se desea asignar a la coordenada X indicada.
4. `dword ValueY`
Valor que se desea asignar a la coordenada Y indicada.

18

```
bool Modify3D(dword CoordinateX,
              dword CoordinateY,
              dword CoordinateZ,
              dword ValueX,
              dword ValueY,
              dword ValueZ)
```

Descripción general

Este método sobrescribe una coordenada de un mapa bidimensional en memoria. **Este método trabaja con lógica negada.**

Retorno

`false`
Si la operación se ha realizado con éxito.
`true`
Si la operación no ha podido ser finalizada.

Parámetros

1. `dword CoordinateX`
Coordenada X de la posición cuyo valor se quiere modificar.
2. `dword CoordinateY`
Coordenada Y de la posición cuyo valor se quiere modificar.
3. `dword CoordinateZ`
Coordenada Z de la posición cuyo valor se quiere modificar.
4. `dword ValueX`
Valor que se desea asignar a la coordenada X indicada.
5. `dword ValueY`
Valor que se desea asignar a la coordenada Y indicada.
6. `dword ValueZ`
Valor que se desea asignar a la coordenada Z indicada.

19

```
bool Position(dword CoordinateX,
              dword CoordinateY,
              dword &PositionX,
              dword &PositionY)
```

Descripción general

Retorna el valor de posición configurado para una coordenada dada en un sistema de dos coordenadas. El valor retornado lleva sumado el Offset. **Este método trabaja con lógica negada.**

Retorno

`false`
Si las coordenadas consultadas existen en el mapa.
`true`
Si las coordenadas consultadas NO existen en el mapa.

Parámetros

1. `dword CoordinateX`
Coordenada X de la posición que se desea consultar.
2. `dword CoordinateY`
Coordenada Y de la posición que se desea consultar.
3. `dword &PositionX`
Valor dentro del mapa de la coordenada X indicada más el offset del mismo.
4. `dword &PositonY`
Valor dentro del mapa de la coordenada Y indicada más el offset del mismo.

20

```
bool Position3D(dword CoordinateX,
                dword CoordinateY,
                dword CoordinateZ,
                dword &PositionX,
                dword &PositionY,
                dword &PositionZ)
```

Descripción general

Retorna el valor de posición configurado para una coordenada dada en un sistema de 3 coordenadas. El valor retornado lleva sumado el Offset. **Este método trabaja con lógica negada.**

Retorno

`false`
Si las coordenadas consultadas existen en el mapa.
`true`
Si las coordenadas consultadas NO existen en el mapa.

Parámetros

1. `dword CoordinateX`
Coordenada X de la posición que se desea consultar.
2. `dword CoordinateY`
Coordenada Y de la posición que se desea consultar.
3. `dword CoordinateZ`
Coordenada Z de la posición que se desea consultar.
4. `dword &PositionX`

Valor dentro del mapa de la coordenada X indicada más el offset del mismo.

5. `dword &PositonY`
   Valor dentro del mapa de la coordenada Y indicada más el offset del mismo.
6. `dword &PositonZ`
   Valor dentro del mapa de la coordenada Z indicada más el offset del mismo.

21

```
bool SetDimension(dword Dimension)
```

Descripción general

Dado que le mismo objeto puede trabajar con varias dimensiones diferentes, este método debe ser llamado durante la inicialización, al menos una vez, a objeto de establecer el número de dimensiones con las que se quiere que el mapa trabaje. **Este método trabaja con lógica negada.**

Retorno

`false`
Si se pudo establecer con éxito ese numero de dimensiones en el mapa
`true`
Si algún fallo impidió establecer el número de dimensiones al indicado (por ejemplo, si se pasa un número menor de 1 como dimensión)

Parámetros

1. `dword Dimension`
   Número de dimensiones que tiene el mapa.

22

```
bool SetSearchRange(dword MaximumX,
                    dword MaximumY)
```

Descripción general

Este método sirve para establecer unas distancias máximas a partir de las cuales no se quiere seguir buscando un punto como más cercano a otro dado. El hecho de establecer estos límites puede acelerar mucho el proceso que se desencadena cada vez que se llama al método GetNearestPosition.
El criterio que se usa es: si cualquiera de las coordenadas de una posición dada está a más distancia de otra (origen) que el máximo establecido, esa posición no se tendrá en cuenta para la búsquedas de la posición más cercana a dicho origen. **Este método trabaja con lógica negada.**

Retorno

`false`
Si la operación se realizó con éxito y se establecieron los parámetros pasados como mínimos de búsqueda.
`true`
Si los parámetros pasados no se pudieron establecer como mínimos. Nótese que es posible establecer valores 0 ó incluso negativo, y no por ello retorna un error (se ha de ser cuidadoso en este punto).

Parámetros

1. `dword MaximumX`
Valor a establecer como distancia máxima en la búsqueda en el eje X.
2. `dword MaximumY`
Valor a establecer como distancia máxima en la búsqueda en el eje Y.

23

```
bool SetSearchRange3D(dword MaximumX,
                      dword MaximumY,
                      dword MaximumZ)
```

Descripción general

Este método sirve para establecer unas distancias máximas a partir de las cuales no se quiere seguir buscando un punto como más cercano a otro dado. El hecho de establecer estos límites puede acelerar mucho el proceso que se desencadena cada vez que se llama al método GetNearestPosition.
El criterio que se usa es: si cualquiera de las coordenadas de una posición dada está a más distancia de otra (origen) que el máximo establecido, esa posición no se tendrá en cuenta para las búsquedas de la posición más cercana a dicho origen. **Este método trabaja con lógica negada.**

Retorno

`false`
Si la operación se realizó con éxito y se establecieron los parámetros pasados como mínimos de búsqueda.
`true`
Si los parámetros pasados no se pudieron establecer como mínimos. Nótese que es posible establecer valores 0 ó incluso negativo, y no por ello retorna un error (se ha de ser cuidadoso en este punto).

Parámetros

1. `dword MaximumX`
Valor a establecer como distancia máxima en la búsqueda en el eje X.
2. `dword MaximumY`
Valor a establecer como distancia máxima en la búsqueda en el eje Y.
3. `dword MaximumZ`
Valor a establecer como distancia máxima en la búsqueda en el eje Z.

24

```
bool Sync(string File)
```

Descripción general

Este método archiva en disco todos los cambios que se han producido en el mapa en memoria. El salvado se realiza de forma **asíncrona**, es decir, pueden pasar varios ciclos entre que se envía la petición hasta que el mapa esté completamente salvado en disco. **Este método trabaja con lógica negada.**

Este método ha cambiado su lógica desde la versión 205 de Galileo 3.1 (05/ 2012), su código en la transición es:
- Versiones anteriores a la 205:
return (this.p_C_TablaPos -> Sync (this.p_IniPosTraslo));
- Versiones posteriores a la 205:
return !(this.p_C_TablaPos -> Sync (this.p_IniPosTraslo));

Retorno

`false`
Si la operación se ha realizado con éxito.
`true`
Si la operación no ha podido ser finalizada (el archivo no ha sido cargado).

Parámetros

1. `string` File
Nombre del fichero en disco donde se quiere salvar el mapa.

25

```
void CountStoredPositions()
```

Descripción general

Devuelve el número de posiciones almacenadas en el fichero.

Retorno

El número de posiciones almacenadas y cargadas del fichero.

Parámetros

Este método no requiere parámetros de entrada.

4.1.2.2 Consideraciones acerca de la eficiencia del componente DimensionalMap

El componente DimensionalMap ha sido diseñado para realizar de manera óptima, en el menos tiempo posible, operaciones que de otro modo pudieran conllevar un coste inaceptable. En concreto, la operación principal del componente DimensionalMap es obtener, a partir de una posición física, la posición lógica correspondiente. Para ello, internamente mantiene una caché con la que minimiza el tiempo necesario para realizar esta operación. Aunque esta caché se mantiene actualizada de manera automática, es posible, que bajo ciertas circunstancias no lo esté, en concreto:

1. Durante la carga del fichero, a fin de evitar actualizar la caché con cada elemento, se actualiza únicamente al final de la carga.
2. Tras una operación BeginUpdate se desactiva la actualización de la caché.

Bajo estas condiciones, para una determinada coordenada física puede no se posible encontrar la coordenada lógica correspondiente. Por ello, a fin de evitar problemas por este funcionamiento, el componente, sino es puede utilizar el método óptimo, utiliza un método que asegura la obtención de la cota lógica, aunque sea en un tiempo mayor. Es en este punto donde tiene sentido la utilización de las funciones SetSearchRange y SetSearchRange3D, que limitan el espació de búsqueda, filtrando las posiciones con distancia en cada eje superior a la fijada. Es importante tener en cuenta que en condiciones normales, el componente no utiliza los limites fijados por estas funciones, ya que utiliza un método óptimo de búsqueda, y solo bajo los supuestos antes enumerados, el fijar un limite de búsqueda es útil.

4.1.2.3 Consideraciones acerca de la modificación de datos del archivo

El componente DimensionalMap permite que el usuario añada, modifique o elimine datos del archivo .ini. Estos datos se mantendrán en memoria hasta que se finalice la ejecución de Galileo, momento en el que se escribirán a disco. A fin de que el programa de control pueda forzar la escritura de los datos en disco, está disponible la operación Sync, que fuerza la escritura de los datos. Sin embargo, esta operación

no es inmediata, ya que las operaciones de E/S son relativamente lentas y podrían ralentizar el ciclo de Galileo de forma peligrosa. A fin evitar este problema, el componente realiza esta operación de forma asíncrona al programa. Esto puede producir a su vez que el programa de control consulte o intente realizar otra operación de escritura mientras la anterior aun está en curso. A fin de control esto, las operaciones que se pueden ver afectadas devuelven un error en este, por lo que el programa de control lo puede detectar para repetir la operación. Otra manera, mas eficiente de realizar este control, es utilizar la operación IOFinished que devuelve trae en caso de que se esté realizando un operación en ese momento o false en caso contrario. De esta manera, el programa de control puede evitar realizar operación cuando hay otras en curso.

Por otro lado, los datos del archivo pueden ser modificados de manera externa a Galileo. En este caso, si el componente ha sido inicializado con el método LoadAndTrack, se realizará una monitorización continua del archivo para comprobar los cambios, y en caso de haberlos se realizará la recarga del fichero. Esto implica que tras un cambio en el fichero, existirá un tiempo en el cual no todos los datos estarán disponibles (se estarán recargando), por lo que es necesario que el programa de control lo tenga en cuenta.

Otro punto importante a tener en cuenta es que durante una operación de modificación iniciada por el programa de control se desactivará la monitorización del fichero, de manera que Galileo no detecte un cambio y recargue de nuevo el fichero. En este punto, si se realiza una operación de modificación externa al mismo tiempo que una desde el programa de control, tendrá prioridad la realizada desde el programa de control.

4.1.3 DIMENSIONALMAPEX

4.1.3.1 Interface

Este componente es una extensión del `DimensionalMap` comentado en el apartado anterior. Tiene todos los métodos de `DimensionMap`, pero además, añade dos que permiten obtener las posiciones físicas adyacentes a una dada.

El interface que presenta este componente es:

Class	`DimensionalMapeX`
BUS IN	0 bytes
BUS OUT	0 bytes

Listado de métodos nuevos (además de los de `DimensionMap`):

1	`bool GetPhysPositionFollowingGradient(dword CurrentPositionX, dword CurrentPositionY, dword DeltaX, dword DeltaY, dword DepthX, dword DepthY,`

	dword &PosX, dword &PosY)
2	bool GetPhysPositionFollowingGradient3D(dword CurrentPositionX, dword CurrentPositionY, dword CurrentPositionZ, dword DeltaX, dword DeltaY, dword DeltaZ, dword DepthX, dword DepthY, dword DepthZ, dword &PosX, dword &PosY, dword &PosZ)

Descripción de métodos nuevos:

1

```
bool GetPhysPositionFollowingGradient(dword CurrentPositionX,
                                      dword CurrentPositionY,
                                      dword DeltaX,
                                      dword DeltaY,
                                      dword DepthX,
                                      dword DepthY,
                                      dword &PosX,
                                      dword &PosY)
```

Descripción general

Dado un mapa bidimensional, mediante este método se permite obtener las coordenadas de la posición a partir de una dada, siguiendo un vector de gradiente también dado.

Retorno

`true`

Si la operación se realizó con éxito, y en los parámetros `PosX` y `PosY` se han almacenado las coordenadas obtenidas.

`false`

Si no se han podido obtener las coordenadas pedidas.

Parámetros

1. `dword CurrentPositionX`
   Coordenada X física de la posición actual.
2. `dword CurrentPositionY`
   Coordenada Y física de la posición actual.
3. `dword DeltaX`
   Valor del vector gradiente en X a utilizar. Realmente solo se utilizará su signo:
   a. Negativo → busca coordenadas físicas menores.
   b. Positivo → busca coordenads físicas mayores.
   c. 0 → no busca en el eje X
4. `dword DeltaY`
   Valor del vector gradiente en Y a utilizar. Realmente solo se utilizará su signo:
   a. Negativo → busca coordenadas físicas menores.

b. Positivo → busca coordenadas físicas mayores.
c. 0 → no busca en el eje Y

5. `dword DepthX`
Profundidad máxima en el eje X en la que realizar la búsqueda.
6. `dword DepthY`
Profundidad máxima en el eje Y en la que realizar la búsqueda.
7. `dword PosX`
Coordenada X encontrada a partir del gradiente indicado en el parámetro `DeltaX`, siguiendo la coordenada indicada en el parámetro `CurrentPositionX`.
8. `dword PosY`
Coordenada Y encontrada a partir del gradiente indicado en el parámetro `DeltaY`, siguiendo la coordenada indicada en el parámetro `CurrentPositionY`.

2

```
bool GetPhysPositionFollowingGradient3D(dword CurrentPositionX,
                                        dword CurrentPositionY,
                                        dword CurrentPositionZ,
                                        dword DeltaX,
                                        dword DeltaY,
                                        dword DeltaZ,
                                        dword DepthX,
                                        dword DepthY,
                                        dword DepthZ,
                                        dword &PosX,
                                        dword &PosY,
                                        dword &PosZ)
```

Descripción general

Dado un mapa tridimensional, mediante este método se permite obtener las coordenadas de la posición a partir de una dada, siguiendo un vector de gradiente también dado.

Retorno

`true`
Si la operación se realizó con éxito, y en los parámetros `PosX` y `PosY` se han almacenado las coordenadas obtenidas.
`false`
Si no se han podido obtener las coordenadas pedidas.

Parámetros

1. `dword CurrentPositionX`
Coordenada X física de la posición actual.
2. `dword CurrentPositionY`
Coordenada Y física de la posición actual.
3. `dword CurrentPositionZ`
Coordenada Z física de la posición actual.
4. `dword DeltaX`
Valor del vector gradiente en X a utilizar. Realmente solo se utilizará su signo:
   d. Negativo → busca coordenadas físicas menores.
   e. Positivo → busca coordenads físicas mayores.
   f. 0 → no busca en el eje X
5. `dword DeltaY`
Valor del vector gradiente en Y a utilizar. Realmente solo se utilizará su signo:

a. Negativo → busca coordenadas físicas menores.
b. Positivo → busca coordenadas físicas mayores.
c. 0 → no busca en el eje Y

6. `dword DeltaZ`
Valor del vector gradiente en Z a utilizar. Realmente solo se utilizará su signo:
a. Negativo → busca coordenadas físicas menores.
b. Positivo → busca coordenads físicas mayores.
c. 0 → no busca en el eje X
7. `dword DepthX`
Profundidad máxima en el eje X en la que realizar la búsqueda.
8. `dword DepthY`
Profundidad máxima en el eje Y en la que realizar la búsqueda.
9. `dword DepthZ`
Profundidad máxima en el eje Z en la que realizar la búsqueda.
10. `dword PosX`
Coordenada X encontrada a partir del gradiente indicado en el parámetro `DeltaX`, siguiendo la coordenada indicada en el parámetro `CurrentPositionX`.
11. `dword PosY`
Coordenada Y encontrada a partir del gradiente indicado en el parámetro `DeltaY`, siguiendo la coordenada indicada en el parámetro `CurrentPositionY`.
12. `dword PosZ`
Coordenada Y encontrada a partir del gradiente indicado en el parámetro `DeltaZ`, siguiendo la coordenada indicada en el parámetro `CurrentPositionZ`.

4.1.4 MACHINE

4.1.4.1 Interface

El componente Machine es el componente más importante del sistema. Permite el acceso a algunas de las propiedades y métodos internos el sistema de ejecución de secuenciadores. Este componente incluye una buena cantidad de métodos, por lo cual presenta cierta complicación entender todas sus posibilidades. En este punto la descripción de todos los métodos sirven como referencia. Una explicación conceptual del trabajo que realiza este componente puede encontrarse en la documentación.

Este componente expone el siguiente interface:

Class	Machine
BUS IN	0 bytes
BUS OUT	0 bytes

Listado de métodos:

1	`void ClearAllTrackings()`
2	`void ClearEventData()`
3	`void ClearIISData()`
4	`void ClearTracking(word TrackingNumber)`
5	`void CopyTracking(byte Tracking1,`

	byte Tracking2, bool Clear)
6	void EvalFailure(dword FailureNumber, bool FailureCond, bool &SignalFail)
7	bool EvalTimer(dword TempNumber, bool Cond)
8	void ExitNotifyEvent()
9	void ExitNotifyIIS()
10	void ExitReceiveCommand()
11	void ExitTerminateCommand()
12	dword GetCurrentStep()
13	byte GetEventData(byte Position)
14	byte GetHeightType()
15	dword GetNextNumber(dword PostNumber)
16	dword GetPrevNumber(dword AntNumber)
17	byte GetSourceAisle()
18	byte GetSourceDepth()
19	byte GetSourceLocalX()
20	byte GetSourceLocalY()
21	byte GetSourceNumber()
22	byte GetSourceSide()
23	byte GetSourceType()
24	word GetSourceX()
25	word GetSourceY()
26	byte GetTargetAisle()
27	byte GetTargetDepth()
28	byte GetTargetLocalX()
29	byte GetTargetLocalY()
30	byte GetTargetNumber()
31	byte GetTargetSide()
32	byte GetTargetType()
33	word GetTargetX()
34	word GetTargetY()
35	dword GetTimeoutError()
36	dword GetTimerCount(dword TempNumber)
37	void GetTracking(word Machine, byte Tracking, bool Clear)
38	byte GetTrackingData(byte Position)
39	dword GetTransportNumber()
40	dword GetVarValue(string Variable)
41	dword GetWTUNumber()
42	dword GetWTUNumberByTracking(dword TrackingNumber)
43	byte GetWTUType()
44	void HoldTimeoutCountdown(bool Hold)
45	void InterchangeTracking(byte Tracking1, byte Tracking2)
46	bool IsCommandReceived()
47	bool IsCommandTerminated()
48	bool IsErrorActive(dword Number)

49	bool IsEventFlagReceived()
50	void IsIISFlagReceived()
51	bool IsRemote()
52	void NotifyEvent(byte EventType)
53	void NotifyIIS()
54	dword Number()
55	void PrepareCommandToFinish(byte Tracking, dword Error, dword AuxData)
56	void ReceiveCommand()
57	void ResetFailure(dword Failure)
58	void ResetFailures()
59	void ResetTimeoutError()
60	void ResetTimer(dword TempNumber)
61	void ResetToAlias(string Alias)
62	void ResetToStep(dword StepNumber)
63	void Restart()
64	void RestartTimer(dword TempNumber)
65	void SelectTracking(dword Number)
66	void SetCurrentAisle(byte Aisle)
67	void SetCurrentAux(byte Position, byte Value)
68	void SetCurrentCapacity(byte Capacity)
69	void SetCurrentOccupation(byte Occupation)
70	void SetCurrentSide(byte Side)
71	void SetCurrentState(byte State)
72	void SetCurrentStationNumber(byte StationNumber)
73	void SetCurrentStationType(byte StationType)
74	void SetCurrentTransport(dword Transport)
75	void SetCurrentX(word X)
76	void SetCurrentY(word Y)
77	void SetEventAux(byte Index, byte Value)
78	void SetEventData(byte Position, byte Value)
79	void SetEventFlags(dword Flags)
80	void SetEventHeightType(byte HeightType)
81	void SetEventPlatformType(byte PlatformType)
82	void SetEventSide(byte Side)
83	void SetEventStationNumber(byte Number)
84	void SetEventStationType(byte Type)
85	void SetEventTransport(dword Transport)
86	void SetEventType(byte Type)
87	void SetEventWeight(dword Weight)
88	void SetEventX(byte X)
89	void SetEventY(byte Y)
90	void SetFailure(dword Failure)
91	void SetFinishAuxData(byte Index, byte Value)
92	void SetFinishHeightType(byte Type)

93	void SetFinishLocalX(byte X)
94	void SetFinishLocalY(byte Y)
95	void SetFinishPlatformType(byte Type)
96	void SetFinishRealStationNumber(byte Number)
97	void SetFinishRealStationType(byte Type)
98	void SetFinishSide(byte Side)
99	void SetFlagsIIS(dword Flags)
100	void SetHeightType(byte HeightType)
101	void SetIISAux(byte Index, byte Value)
102	void SetIISData(byte Position, byte Value)
103	void SetIISHeightType(byte HeightType)
104	void SetIISPlatformType(byte PlatformType)
105	void SetIISStationNumber(dword Number)
106	void SetIISStationType(dword Type)
107	void SetIISTransport(dword Transport)
108	void SetIISWeight(dword Weight)
109	void SetSelector(dword SelectNumber)
110	void SetSourceAisle(byte SourceAisle)
111	void SetSourceDepth(byte SourceDepth)
112	void SetSourceLocalX(byte X)
113	void SetSourceLocalY(byte Y)
114	void SetSourceNumber(byte SourceNumber)
115	void SetSourceSide(byte SourceSide)
116	void SetSourceType(byte SourceType)
117	void SetSourceX(word SourceX)
118	void SetSourceY(word SourceX)
119	void SetStationStatus(byte StationType, byte StationNumber, byte Status, byte Loaded, byte Capacity, byte CurrentCount, byte Aisle)
120	void SetTargetAisle(byte TargetAisle)
121	void SetTargetDepth(byte TargetDepth)
122	void SetTargetLocalX(byte X)
123	void SetTargetLocalY(byte Y)
124	void SetTargetNumber(byte TargetNumber)
125	void SetTargetSide(word TargetSide)
126	void SetTargetType(byte TargetType)
127	void SetTargetX(word TargetX)
128	void SetTargetY(word TargetY)
129	void SetTimerSA(dword TempNumber, dword Tempmseg)
130	void SetTimerSE(dword TempNumber, dword Tempmseg)
131	void SetTimerSI(dword TempNumber, dword Tempmseg)
132	void SetTimerSS(dword TempNumber,

	dword Tempmseg)
133	void SetTimerSV(dword TempNumber, dword Tempmseg)
134	void SetTrackingData(byte Position, byte Value)
135	void SetTransportNumber(dword Number)
136	void SetVarValue(string Variable, dword Value)
137	void SetWTUNumber(dword WTUNumber)
138	void SetWTUType(byte WTUType)
139	void SimulateSignal(bool &SimulateSignal, bool StartCond, bool StopCond, dword ActivateTime, dword DesactivateTime)
140	void TerminateCommand()
141	dword TrackingCount()
142	word GetPriority(void)
143	word GetSequence(void)
144	word GetSpeedX(void)
145	word GetSpeedY(void)
146	word GetSpeedZ(void)
147	void SetPriority(word value)
148	void SetSequence(word value)
149	void SetSpeedX(word value)
150	void SetSpeedY(word value)
151	void SetSpeedZ(word value)
152	void SetIISLocalX(word value)
153	void SetIISLocalY(word value)
154	void SetIISDepth(byte value)
155	void SetIISAisle(byte value)
156	void SetMultiOPIISFlags(byte slot, dword value)
157	void SetMultiOPIISStationNumber(byte slot, byte value)
158	void SetMultiOPIISStationType(byte slot, byte value)
159	void SetMultiOPIISAuxData(byte slot, byte index, byte value)
160	byte GetMultiOPIISAuxData(byte slot, byte index)
161	void SetMultiOPIISTransport(byte slot, dword transport)
162	void SetMultiOPIISWeight(byte slot, dword weight)
163	void SetMultiOPIISPlatformType(byte slot, byte platformType)
164	void SetMultiOPIISHeightType(byte slot, byte height)
165	void SetMultiOPIISX(byte slot, word x)
166	void SetMultiOPIISY(byte slot, word y)
167	void SetMultiOPIISSide(byte slot, byte side)
168	void SetMultiOPIISEventType(byte slot, byte eventType)
169	void SetMultiOPIISLocalX(byte slot, byte localX)
170	void SetMultiOPIISLocalY(byte slot, byte localY)
171	void SetMultiOPIISDepth(byte slot, byte depth)
172	void SetMultiOPIISAisle(byte slot, byte aisle)
173	void SetMultiOPIISAux(byte slot, byte index, byte value)
174	void ClearMultiOPIISData(byte slot)
175	void SetMultiOPEndRealStationNumber(byte slot, byte number)

176	void SetMultiOPEndRealStationType(byte slot, byte type)
177	void SetMultiOPEndPlatformType(byte slot, byte platformType)
178	void SetMultiOPEndHeightType(byte slot, byte heightType)
179	void SetMultiOPEndLocalX(byte slot, byte localX)
180	void SetMultiOPEndLocalY(byte slot, byte localY)
181	void SetMultiOPEndSide(byte slot, byte side)
182	void SetMultiOPEndAux(byte slot, byte index, byte value)
183	void SetMultiOPEndLogicalX(byte slot, word logicalX)
184	void SetMultiOPEndLogicalY(byte slot, word logicalY)
185	void SetMultiOPEndDepth(byte slot, byte depth)
186	void SetMultiOPEndAisle(byte slot, byte aisle)
187	void PrepapeMultiOPCommandToFinish(byte slot, byte tracking, dword errorCode, dword aux)
188	void SetMultiOPSearchCurrentX(word X)
189	void SetMultiOPSearchCurrentY(word Y)
190	void SetMultiOPSearchCurrentSide(byte side)
191	void SetMultiOPSearchCurrentAisle(byte aisle)
192	void SetMultiOPSearchTransport(dword transport)
193	void SetMultiOPSearchCapacity(byte capacity)
194	void SetMultiOPSearchOccupation(byte occupation)
195	void SetMultiOPSearchStationNumber(byte stationNumber)
196	void SetMultiOPSearchStationType(byte stationType)
197	void SetMultiOPSearchAuxData(byte index, byte value)
198	void ExecuteMultiOP(bool searchActive, byte endCounter, byte iisCounter)
199	bool IsMultiOPFinished(void)
200	void ExitMultiOP(void)

Descripción de métodos:

1

```
void ClearAllTrackings()
```

Descripción general

Este método sirve para borrar los datos de todos los trackings almacenados en la máquina.

Retorno

void

Este método no devuelve nada

Parámetros

Este método no tiene parámetros de entrada.

2

```
void ClearEventData()
```

Descripción general

Este método limpia el buffer de almacenamiento para los datos del evento. Se utilizará para limpiar después de haber notificado.

Retorno

`void`

Este método no retorna nada.

Parámetros

Este método no tiene parámetros de entrada.

3

```
void ClearIISData()
```

Descripción general

Este método limpia el buffer de almacenamiento para los datos del Palet en PIE. Se utilizará para limpiar después de haber notificado.

Retorno

`void`

Este método no retorna nada.

Parámetros

Este método no tiene parámetros de entrada.

4

```
void ClearTracking(word TrackingNumber)
```

Descripción general

Este método sirve para borrar los datos de un tracking (rellena con 0). Nótese que esto elimina el tracking, pero si este estaba seleccionado, aun se puede

Retorno

`void`

Este método no devuelve nada

Parámetros

1. `word` `TrackingNumber`

   Número de tracking que se desea borrar

5

```
void CopyTracking(byte Tracking1,
                  byte Tracking2,
                  bool Clear)
```

Descripción general

Este método copia un tracking de una posición a otra dentro de la misma máquina. Tiene además la opción de borrar el Tracking de Origen.

Retorno

`void`

Este método no retorna nada

Parámetros

1. `byte` `Tracking1`

   Tracking destino de la copia

2. byte Tracking2
Tracking origen de la copia
3. bool Clear
Variable que representa si se desea borrar el tracking origen:
    a. true → Borrar tracking origen
    b. false → NO borrar tracking origen

6

```
void EvalFailure(dword FailureNumber,
                 bool FailureCond,
                 bool &SignalFail)
```

Descripción general

Este método evalúa la posibilidad de la existencia de una avería. Gestiona internamente la señalización a la visualización. Este es el método recomendado para la gestión de averías.

Retorno

void
Este método no retorna nada

Parámetros

1. dword FailureNumber
Número de fallo a evaluar
2. bool FailureCond
Señal booleana de avería. Esta señal se interpreta con lógica negativa, es decir, si la condición de avería está a true, no existe avería.
3. bool SignalFail
Variable booleana pasada por referencia. Esta variable se modifica para señalar avería si se da la condición de avería, en cuyo caso pasará a tener el valor true. Si no hay condición de avería la variable no modifica su valor.

7

```
bool EvalTimer(dword TempNumber,
               bool Cond)
```

Descripción general

Evalua si se ha producido la condición de un Timer.

Retorno

void
Este método no retorna nada

Parámetros

1. dword TempNumber
Número de temporizador que se desea utilizar, dentro del rango de 0 a 254.
2. bool Cond
Resultado de la condición que arranca el temporizador.

8

```
void ExitNotifyEvent()
```

Descripción general

Este método finaliza el protocolo de notificación del evento al Agente de Transportes.

Retorno

void

Este método no retorna nada.

Parámetros

Este método no tiene parámetros de entrada.

9

```
void ExitNotifyIIS()
```

Descripción general

Este método finaliza el protocolo de notificación de Palet en PIE al Agente de Transportes.

Retorno

void

Este método no retorna nada.

Parámetros

Este método no tiene parámetros de entrada.

10

```
void ExitReceiveCommand()
```

Descripción general

Señaliza al Agente de Transportes que esta máquina finaliza el protocolo de recepción de orden.

Retorno

void

Este método no retorna nada.

Parámetros

Este método no tiene parámetros de entrada.

11

```
void ExitTerminateCommand()
```

Descripción general

Este método finaliza el protocolo de comunicaciones de finalización de orden con el Agente de Transportes.

Retorno

void

Este método no retorna nada

Parámetros

Este método no tiene parámetros de entrada

12

```
dword GetCurrentStep()
```

Descripción general

Este método sirve para obtener el número de etapa activa actualmente en la máquina. Nótese que puede haber más de una etapa activa en función del grafcet de la máquina. En ese caso solo se devuelve la primera de ellas.

Retorno

dword

Indica el número de la primera etapa activa en la máquina en ese momento.

Parámetros

Este método no tiene parámetros de entrada.

13

```
byte GetEventData(byte Position)
```

Descripción general

Este método devuelve los datos almacenados en el PIE.

Retorno

byte

Retorna el valor almacenado en la posición indicada mediante el parámetro Position.

Parámetros

1. byte Position
   Lugar en el cual se encuentra el dato, con un rango de 0 a 255.

14

```
byte GetHeightType()
```

Descripción general

Este método sirve para devolver el tipo de la altura del tracking seleccionado (que por defecto es 1).

Retorno

byte

Retorna el tipo de altura.

Parámetros

Este método no tiene parámetros de entrada

15

```
dword GetNextNumber(dword PostNumber)
```

Descripción general

Este método se emplea para poder consultar el número de máquina que se encuentra ubicada como posterior 1, posterior 2, etc.

Retorno

dword

Retorna el número de máquina configurada en dicha posición de posterior. Retorna valor 0 si no existe ninguna máquina configurada como la posterior indicada

Parámetros

1. dword PostNumber
   Posición de posterior que se desea consultar (posterior 1 sería 1, posterior 2 sería 2, etc.).

16

```
dword GetPrevNumber(dword AntNumber)
```

Descripción general

Este método se emplea para poder consultar el número de máquina que se encuentra ubicada como anterior 1, anterior 2, etc.

Retorno

dword
Retorna el número de máquina configurada en dicha posición de anterior. Retorna valor 0 si no existe ninguna máquina configurada como la anterior indicada

Parámetros

1. dword AntNumber
   Posición de anterior que se desea consultar (anterior 1 sería 1, anterior 2 sería 2, etc.).

17

```
byte GetSourceAisle()
```

Descripción general

Este método sirve para devolver el pasillo de origen del tracking seleccionado (que por defecto es 1).

Retorno

byte
Retorna el pasillo de origen del tracking

Parámetros

Este método no tiene parámetros de entrada

18

```
byte GetSourceDepth()
```

Descripción general

Este método sirve para devolver la profundidad de origen del tracking seleccionado (que por defecto es 1).

Retorno

byte
Retorna la profundidad de origen del tracking

Parámetros

Este método no tiene parámetros de entrada

19

```
byte GetSourceLocalX()
```

Descripción general

Este método devuelve la coordenada X (origen) del tracking seleccionado.

Retorno

byte
Retorna la coordenada X de origen.

Parámetros

Este método no tiene parámetros de entrada.

20

```
byte GetSourceLocalY()
```

Descripción general

Este método devuelve la coordenada Y (origen) del tracking seleccionado.

Retorno

byte
Retorna la coordenada Y de origen.

Parámetros

Este método no tiene parámetros de entrada.

21

```
byte GetSourceNumber()
```

Descripción general

Este método sirve para devolver el número de la entidad origen del tracking seleccionado (que por defecto es 1).

Retorno

byte
Retorna el número de entidad origen.

Parámetros

Este método no tiene parámetros de entrada

22

```
byte GetSourceSide()
```

Descripción general

Este método sirve para devolver el lado de origen del tracking seleccionado (que por defecto es 1).

Retorno

byte
Retorna el lado de origen del tracking

Parámetros

Este método no tiene parámetros de entrada

23

```
byte GetSourceType()
```

Descripción general

Este método sirve para devolver el tipo de la entidad origen del tracking seleccionado (que por defecto es 1).

Retorno

byte

Retorna el tipo de entidad origen.

Parámetros

Este método no tiene parámetros de entrada

24

```
word GetSourceX()
```

Descripción general

Este método sirve para devolver la coordenada X de origen del tracking seleccionado (que por defecto es 1).

Retorno

word

Retorna la coordenada X del origen del tracking

Parámetros

Este método no tiene parámetros de entrada

25

```
word GetSourceY()
```

Descripción general

Este método sirve para devolver la coordenada Y de origen del tracking seleccionado (que por defecto es 1).

Retorno

word

Retorna la coordenada Y del origen del tracking

Parámetros

Este método no tiene parámetros de entrada

26

```
byte GetTargetAisle()
```

Descripción general

Este método sirve para devolver el pasillo de destino del tracking seleccionado (que por defecto es 1).

Retorno

byte

Retorna el pasillo de destino del tracking

Parámetros

Este método no tiene parámetros de entrada

27

```
byte GetTargetDepth()
```

Descripción general

Este método sirve para devolver la profundidad de destino del tracking seleccionado (que por defecto es 1).

Retorno

byte

Retorna la profundidad de destino del tracking

Parámetros

Este método no tiene parámetros de entrada

28

```
byte GetTargetLocalX()
```

Descripción general

Este método devuelve la coordenada X (destino) del tracking seleccionado.

Retorno

byte

Retorna la coordenada X de destino.

Parámetros

Este método no tiene parámetros de entrada.

29

```
byte GetTargetLocalY()
```

Descripción general

Este método devuelve la coordenada Y (destino) del tracking seleccionado.

Retorno

byte

Retorna la coordenada Y de destino.

Parámetros

Este método no tiene parámetros de entrada.

30

```
byte GetTargetNumber()
```

Descripción general

Este método sirve para devolver el número de la entidad destino del tracking seleccionado (que por defecto es 1).

Retorno

byte

Retorna el número de la entidad destino del tracking.

Parámetros

Este método no tiene parámetros de entrada

31

```
byte GetTargetSide()
```

Descripción general

Este método sirve para devolver el lado de destino del tracking seleccionado (que por defecto es 1).

Retorno

byte

Retorna el lado de destino del tracking

Parámetros

Este método no tiene parámetros de entrada

32

```
byte GetTargetType()
```

Descripción general

Este método sirve para devolver el tipo de la entidad destino del tracking seleccionado (que por defecto es 1).

Retorno

byte

Retorna el tipo de la entidad destino del tracking.

Parámetros

Este método no tiene parámetros de entrada

33

```
word GetTargetX()
```

Descripción general

Este método sirve para devolver la coordenada X de destino del tracking seleccionado (que por defecto es 1).

Retorno

word

Retorna la coordenada X de destino del tracking

Parámetros

Este método no tiene parámetros de entrada

34

```
word GetTargetY()
```

Descripción general

Este método sirve para devolver la coordenada Y de destino del tracking seleccionado (que por defecto es 1).

Retorno

word

Retorna la coordenada Y de destino del tracking

Parámetros

Este método no tiene parámetros de entrada

35

```
dword GetTimeoutError()
```

Descripción general

Este método devuelve el número de error especificado en caso de que se haya producido un timeout en alguna de las etapas de la máquina.

Retorno

`dword`

Número de error especificado como *Timeout* en la etapa donde se ha producido.

Parámetros

Este método no tienen parámetros de entrada

36

```
dword GetTimerCount(dword TempNumber)
```

Descripción general

Este método retorna la cuenta actual del temporizador indicado. Recordemos que dicha cuenta está en milisegundos y que es una cuenta descendente.

Retorno

`dword`

Retorna el valor actual de la cuenta del temporizador.

Parámetros

1. `dword TempNumber`

   Número de temporizador que se desea utilizar, dentro del rango de 0 a 254.

37

```
void GetTracking(word Machine,
                 byte Tracking,
                 bool Clear)
```

Descripción general

Este método copia el tracking 1 de la máquina deseada a la posición de tracking seleccionada por nosotros. Se permite eliminar el tracking origen.

Retorno

`void`

Este método no retorna nada

Parámetros

1. `word Machine`

   Máquina de la cual se desea adquirir el tracking
2. `byte Tracking`

   Número de tracking en el que se desea recibir el trakcing copiado
3. `bool Clear`

   Variable que representa si se desea borrar el tracking origen:
   a. `true` → Borrar tracking origen y desplazar la pila de trackings para reorganizar de nuevo la información.
   b. `false` → NO borrar tracking origen

38

```
byte GetTrackingData(byte Position)
```

Descripción general

La estructura del tracking posee un área de datos utilizables por el usuario. Este método permite leer esta área. Hay que hacer notar que estos datos son arrastrados con el tracking.

Retorno

byte

Contenido de la posición especificada

Parámetros

1. byte Position

   Posición del área de datos de la cual se desea leer. Los valores posibles van desde 0 a 49, que corresponde con el área de estos datos de tracking reservada al Sistema de Control.

39

```
dword GetTransportNumber()
```

Descripción general

Este método sirve para devolver el número de transporte del tracking seleccionado (que por defecto es 1).

Retorno

dword

Número de transporte

Parámetros

Este método no tiene parámetros de entrada

40

```
dword GetVarValue(string Variable)
```

Descripción general

Este método realiza una consulta indirecta del valor de una variable. Solamente realiza consultas de los valores de las variables de dicha máquina.

Retorno

dword

Retorna el valor de la variable consultada. En caso de que dicha variable no exista, retorna siempre el valor 0.

Parámetros

1. string Variable

   Nombre de la variable que se desea consultar.

41

```
dword GetWTUNumber()
```

Descripción general

Este método sirve para devolver el número de la unidad de transporte del tracking seleccionado (que por defecto es 1).

Retorno

dword

Retorna el número de la unidad de transporte. Debe hacerse notar que internamente la unidad de transporte es un entero de 64 bits. En el caso de este método retorna un entero de 32 bits con lo cual en caso de no entrar sería recortado.

Parámetros

Este método no tiene parámetros de entrada

42

```
dword GetWTUNumberByTracking(dword TrackingNumber)
```

Descripción general

Este método devuelve el valor de UMA (WTU) del tracking que se le pasa como parámetro, aunque éste no este seleccionado.

Retorno

dword

Devuelve la UMA del tracking seleccionado

Parámetros

1. dword TrackingNumber
   Número de tracking del que se desea consulta la UMA

43

```
byte GetWTUType()
```

Descripción general

Este método sirve para devolver el tipo de la unidad de transporte del tracking seleccionado (que por defecto es 1).

Retorno

byte

Retorna el tipo de la unidad de transporte

Parámetros

Este método no tiene parámetros de entrada

44

```
void HoldTimeoutCountdown(bool Hold)
```

Descripción general

Este método se emplea para poder suspender durante el ciclo actual la cuenta de "Timeout" de la etapa desde la que se invoca. La invocación de este método no tiene ningún efecto si la etapa activa no tiene definido un valor de timeout.

Retorno

void

Este método no retorna nada.

Parámetros

1. `bool` Hold
Booleano que indica cuando se quiere iniciar la cuenta del timeout configurado en la etapa.
NOTA: Si este método es llamado desde un método que no pertenece a una etapa (la acción anterior del secuenciador, por ejemplo), en lugar de suspender la cuenta de una etapa concreta, intentará suspender la cuenta de todas las etapas activas en el momento de la invocación.

45

```
void InterchangeTracking(byte Tracking1,
                         byte Tracking2)
```

Descripción general

Este método permite intercambiar trackings dentro de la misma máquina.

Retorno

`void`
Este método no retorna nada

Parámetros

1. `byte` `Tracking1`
Tracking destino de la copia
2. `byte` `Tracking2`
Tracking origen de la copia

46

```
bool IsCommandReceived()
```

Descripción general

El Agente de Transportes señala a la máquina si se ha recibido una orden para ella, en cuyo caso se encontrará SIEMPRE en el Tracking número 1.

Retorno

`true`
Si se ha recibido alguna orden para la máquina
`false`
Si NO se ha recibido alguna orden para la máquina

Parámetros

Este método no tiene parámetros de entrada

47

```
bool IsCommandTerminated()
```

Descripción general

Este método sirve para saber si el Agente de Transportes ha reconocido el fin de orden.

Retorno

`true`
Si el Agente de Transportes ha procesado el fin de orden
`false`
Si el Agente de Transportes NO ha procesado el fin de orden

Parámetros

Este método no tiene parámetros de entrada

48

```
bool IsErrorActive(dword Number)
```

Descripción general

Este método sirve para averiguar si un determinado error se encuentra activo en la máquina.

Retorno

`true`
Si el error indicado como parámetro se encuentra activo en ese momento.
`false`
Si el error indicado como parámetro NO se encuentra activo en ese momento.

Parámetros

1. `dword` Number
   Número de error del que se desea conocer si está activo o no.

49

```
bool IsEventFlagReceived()
```

Descripción general

Este método sirve para enterarse de cuando el Agente de Transportes ha procesado el evento.

Retorno

`true`
Si el Agente de Transportes ha procesado el evento.
`false`
Si el Agente de Transportes aún NO ha procesado el evento.

Parámetros

Este método no tiene parámetros de entrada.

50

```
void IsIISFlagReceived()
```

Descripción general

Este método sirve para enterarse de cuando el Agente de Transportes ha procesado el PIE.

Retorno

`true`
Si el Agente de Transportes ha procesado el palet en PIE.
`false`
Si el Agente de Transportes NO ha procesado el palet en PIE.

Parámetros

Este método no tiene parámetros de entrada.

51

```
bool IsRemote()
```

Descripción general

Este método nos indica si una máquina determinada es remota o no. El Sistema Galileo III permite la utilización de programas compartidos entre varios computadores. En general dichos computadores ejecutan cada uno una serie de máquinas, pero en el programa de control pueden realizarse operaciones con componentes Machine remotos. El sistema no soporta realizar determinadas operaciones en remoto. Para poder consultar si una máquina se está ejecutando en un computador diferente del actual se puede utilizar este método.

Retorno

`true`
La máquina se está ejecutando en un computador diferente del actual.
`false`
La máquina NO se está ejecutando en un computador diferente del actual.

Parámetros

Este método no tiene parámetros de entrada.

52

```
void NotifyEvent(byte EventType)
```

Descripción general

Este método sirve para notificar al Agente de Transportes de la existencia de un evento. Los datos transmitidos son los preparados antes con los métodos previos.

Retorno

`void`
Este método no retorna nada.

Parámetros

1. `byte EventType`
   Tipo de evento a notificar:
   a. 1 → Palet en PIE

53

```
void NotifyIIS()
```

Descripción general

Este método sirve para notificar al Agente de Transportes de la existencia de un palet en PIE. Los datos transmitidos son los preparados antes con los métodos previos.

Retorno

`void`
Este método no retorna nada.

Parámetros

Este método no tiene parámetros de entrada.

54

```
dword Number()
```

Descripción general

Este método sirve para interrogar a una máquina sobre su número interno.

Retorno

dword
Número de máquina configurado dentro de la aplicación de desarrollo Designer.

Parámetros

Este método no tiene parámetros de entrada.

55

```
void PrepareCommandToFinish(byte Tracking,
                            dword Error,
                            dword AuxData)
```

Descripción general

Este método prepara un final de orden cargando los datos del tracking para utilizarlos en la orden que se finalizará.

Retorno

void
Este método no retorna nada.

Parámetros

1. byte Tracking
   Número del tracking en el que se encuentra la orden que se desea finalizar
2. dword Error
   Código de error con el que se va a finalizar la orden
3. dword AuxData
   Datos auxiliares que se desean indicar en la finalización.

56

```
void ReceiveCommand()
```

Descripción general

El Agente de Transportes señala a la máquina si se ha recibido una orden para ella, en cuyo caso se encontrará SIEMPRE en el Tracking número 1.

Retorno

void
Este método no retorna ningún valor

Parámetros

Este método no tiene parámetros de entrada

57

```
void ResetFailure(dword Failure)
```

Descripción general

Elimina un fallo en el sistema. El método solamente elimina la avería a efectos de que la visualización la muestre.

Retorno

void
Este método no retorna nada

Parámetros

1. `dword` `Failure`
Número fallo que se desea eliminar.

58

```
void ResetFailures()
```

Descripción general

Este método elimina a efectos de la visualización todas las averías de una máquina. Este método debe asociarse a las acciones de rearme.

Retorno

`void`
Este método no retorna nada

Parámetros

Este método no recibe parámetros de entrada.

59

```
void ResetTimeoutError()
```

Descripción general

Este método sirve para dar un error de timeout por finalizado en la máquina actual. Ha de hacerse notar que el hecho de que se produzca un error por Timeout en alguna etapa no compromete para nada la ejecución del secuenciador de la máquina, y es el propio programador el que debe proporcionar los mecanismos adecuados para actuar cuando este se produce.

Retorno

`void`
Este método no retorna nada.

Parámetros

Este método no tienen parámetros de entrada

60

```
void ResetTimer(dword TempNumber)
```

Descripción general

Resetea el temporizador indicado en el parámetro `TempNumber`.

Retorno

`void`
Este método no retorna nada

Parámetros

1. `dword` `TempNumber`
Número de temporizador que se desea utilizar, dentro del rango de 0 a 254.

61

```
void ResetToAlias(string Alias)
```

Descripción general

Este método realiza un reset a la etapa indicada a través del Alias. Este método es

el preferido para realizar reset desde el programa de control, ya que aisla dichos reset de los números de etapa, dado que estos últimos pueden cambiar si se modifica el grafo.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `string` Alias
   Nombre del alias de la etapa a la que se desea realizar el reset.

62

```
void ResetToStep(dword StepNumber)
```

Descripción general

Este método sirve para forzar el cambio a la etapa cuyo número se pasa como parámetro. La invocación de este método puede bloquear el funcionamiento de la máquina, por ejemplo, si se resetea la máquina a una etapa en medio de una bifurcación AND.

Retorno

`void`

Este método no retorna nada

Parámetros

1. `dword` StepNumber
   Número de etapa a la que se desea resetear la máquina.

63

```
void Restart()
```

Descripción general

Este método resetea el secuenciador.
NOTA: La invocación de este método con la máquina en movimiento puede tener consecuencias desagradables.

Retorno

`void`

Este método no retorna nada

Parámetros

Este método no recibe parámetros de entrada.

64

```
void RestartTimer(dword TempNumber)
```

Descripción general

Reinicia la cuenta del temporizador indicado en el parámetro `TempNumber`.

Retorno

`void`

Este método no retorna nada

Parámetros

1. `dword` TempNumber
   Número de temporizador que se desea utilizar, dentro del rango de 0 a 254.

65

```
void SelectTracking(dword Number)
```

Descripción general

Este método selecciona el tracking con el que se desea trabajar en las funciones que se verán a continuación.

Retorno

`void`

Este método no retorna nada

Parámetros

1. `dword` `Number`

   Número del tracking con el cual queremos trabajar. Sus valores posibles son de 1 a 10. Nada impide que en un futuro se incremente el número de trackings de los que dispone una máquina.

66

```
void SetCurrentAisle(byte Aisle)
```

Descripción general

Este método permite informar al Agente de Transportes del pasillo actual de la máquina para que pueda seleccionar la mejor opción a la hora de buscar orden.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `byte` `Side`

   Pasillo actual de la máquina

67

```
void SetCurrentAux(byte Position,
                   byte Value)
```

Descripción general

Estable el valor de los auxiliares de la búsqueda de orden que se va a realizar.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `byte` `Position`

   Lugar en el cual se archivará el dato.

2. `byte` `Value`

   Valor del dato que se desea almacenar.

68

```
void SetCurrentCapacity(byte Capacity)
```

Descripción general

Establece la capacidad actual en la búsqueda de orden que se va a realizar.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `byte` `Capacity`
   Capacidad actual de la máquina.

69

```
void SetCurrentOccupation(byte Occupation)
```

Descripción general

Establece la ocupación actual en la búsqueda de orden que se va a realizar.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `byte` `Occupation`
   Ocupación actual de la máquina.

70

```
void SetCurrentSide(byte Side)
```

Descripción general

Este método permite informar al Agente de Transportes el lado actual de la máquina para que pueda seleccionar la mejor opción a la hora de buscar orden.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `byte` `Side`
   Lado actual de la máquina

71

```
void SetCurrentState(byte State)
```

Descripción general

Este método permite informar al Agente de Transportes del estado actual de la máquina para que pueda seleccionar la mejor opción a la hora de buscar orden.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `byte` `State`
   Estado actual de la máquina

72

```
void SetCurrentStationNumber(byte StationNumber)
```

Descripción general

Establece el número de estación que realiza la búsqueda de orden que se va a realizar.

Retorno

```
void
```

Este método no retorna nada.

Parámetros

1. `byte StationNumber`
   Número de estación actual.

73

```
void SetCurrentStationType(byte StationType)
```

Descripción general

Establece el tipo de estación que realiza la búsqueda de orden que se va a realizar.

Retorno

```
void
```

Este método no retorna nada.

Parámetros

1. `byte StationType`
   Tipo de estación actual.

74

```
void SetCurrentTransport(dword Transport)
```

Descripción general

Establece el número de transporte en la búsqueda a realizar.

Retorno

```
void
```

Este método no retorna nada.

Parámetros

1. `dword Transport`
   Número de transporte del tracking actual.

75

```
void SetCurrentX(word X)
```

Descripción general

Este método permite informar al Agente de Transportes de la situación X actual de la máquina para que pueda seleccionar la mejor opción a la hora de buscar orden.

Retorno

```
void
```

Este método no retorna nada.

Parámetros

1. `word X`
   Coordenada X actual de la máquina

76

```
void SetCurrentY(word Y)
```

Descripción general

Este método permite informar al Agente de Transportes de la situación Y actual de la máquina para que pueda seleccionar la mejor opción a la hora de buscar orden.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `word` `Y`

   Coordenada Y actual de la máquina

77

```
void SetEventAux(byte Index,
                 byte Value)
```

Descripción general

Este método prepara los datos auxiliares para el evento que se van a notificar al Agente de Transportes.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `byte` `Index`

   Índice del dato a preparar.

2. `byte` `Value`

   Valor que se desea almacenar.

78

```
void SetEventData(byte Position,
                  byte Value)
```

Descripción general

Este método prepara los datos del evento que se van a notificar al Agente de Transportes.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `byte` `Position`

   Lugar en el cual se archivará el dato, con un rango de 0 a 255.

2. `byte` `Value`

   Valor que se desea almacenar.

79

```
void SetEventFlags(dword Flags)
```

Descripción general

Este método prepara los datos un evento. Éste es el flag que se transmitirá al Agente de Transportes cuando se notifique el evento.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. dword Flags
   Valor del flag que se desea forzar en el evento.

80

```
void SetEventHeightType(byte HeightType)
```

Descripción general

Este método prepara los datos del tipo de altura para el evento que se van a notificar al Agente de Transportes.

Retorno

void
Este método no retorna nada.

Parámetros

1. byte HeightType
   Datos del tipo de altura que se informa al evento.

81

```
void SetEventPlatformType(byte PlatformType)
```

Descripción general

Este método prepara los datos del tipo de contenedor para el evento que se van a notificar al Agente de Transportes.

Retorno

void
Este método no retorna nada.

Parámetros

1. byte PlatformType
   Datos del tipo de contenedor que se informa al evento.

82

```
void SetEventSide(byte Side)
```

Descripción general

Este método establece el lado del evento que se va a notificar al Agente de Transportes.

Retorno

void
Este método no retorna nada.

Parámetros

1. byte Side
   Lado que se informa en el evento.

83

```
void SetEventStationNumber(byte Number)
```

Descripción general

Este método prepara el número de la estación que a notificar un evento al Agente de Transportes.

Retorno

void

Este método no retorna nada.

Parámetros

1. byte Number

Número de la estación que notificará el evento.

84

```
void SetEventStationType(byte Type)
```

Descripción general

Este método prepara el tipo de la estación que a notificar un evento al Agente de Transportes.

Retorno

void

Este método no retorna nada.

Parámetros

1. byte Type

Tipo de la estación que notificará el evento.

85

```
void SetEventTransport(dword Transport)
```

Descripción general

Este método prepara los datos del numero de transporte para el evento que se van a notificar al Agente de Transportes.

Retorno

void

Este método no retorna nada.

Parámetros

1. dword Transport

Número de transporte que se informa al evento.

86

```
void SetEventType(byte Type)
```

Descripción general

Este método establece el tipo de evento que se quiere notificar.

Retorno

void

Este método no retorna nada.

Parámetros

1. byte Type

Tipo de evento se desea.

87

```
void SetEventWeight(dword Weight)
```

Descripción general

Este método prepara los datos del peso para el evento que se van a notificar al Agente de Transportes.

Retorno

void

Este método no retorna nada.

Parámetros

1. dword Weight

   Datos del peso que se informa al evento.

88

```
void SetEventX(byte X)
```

Descripción general

Este método establece la coordenada X del evento que se va a notificar al Agente de Transportes.

Retorno

void

Este método no retorna nada.

Parámetros

1. byte X

   Coordenada X que se informa en el evento.

89

```
void SetEventY(byte Y)
```

Descripción general

Este método establece la coordenada Y del evento que se va a notificar al Agente de Transportes.

Retorno

void

Este método no retorna nada.

Parámetros

1. byte Y

   Coordenada Y que se informa en el evento.

90

```
void SetFailure(dword Failure)
```

Descripción general

Establece un fallo en el sistema. El método solamente introduce la avería a efectos de que la visualización la muestre.

Retorno

void

Este método no retorna nada

Parámetros

1. dword Failure

   Número fallo que se desea introducir.

91

```
void SetFinishAuxData(byte Index,
```

```
                    byte Value)
```

Descripción general

Este método permite cargar los datos auxiliares pasados durante la finalización de una orden.

Retorno

```
void
```

Este método no retorna nada.

Parámetros

1. `byte Index`
   Índice del dato a modificar.
2. `byte Value`
   Valor que se desea almacenar.

92

```
void SetFinishHeightType(byte Type)
```

Descripción general

Este método carga el tipo de altura que finaliza la orden.

Retorno

```
void
```

Este método no retorna nada.

Parámetros

1. `byte Type`
   Tipo de altura que realmente finaliza la orden.

93

```
void SetFinishLocalX(byte X)
```

Descripción general

Este método carga la coordenada X en la que se finaliza la orden.

Retorno

```
void
```

Este método no retorna nada.

Parámetros

1. `byte X`
   Coordenada X donde realmente se finaliza la orden.

94

```
void SetFinishLocalY(byte Y)
```

Descripción general

Este método carga la coordenada Y en la que se finaliza la orden.

Retorno

```
void
```

Este método no retorna nada.

Parámetros

1. `byte Y`
   Coordenada Y donde realmente se finaliza la orden.

95

```
void SetFinishPlatformType(byte Type)
```

Descripción general

Este método carga el tipo de contenedor que finaliza la orden.

Retorno

```
void
```

Este método no retorna nada.

Parámetros

1. `byte Type`
   Tipo de contenedor que realmente finaliza la orden.

96

```
void SetFinishRealStationNumber(byte Number)
```

Descripción general

Este método carga el número de estación que realmente finaliza la orden.

Retorno

```
void
```

Este método no retorna nada.

Parámetros

1. `byte Number`
   Número de estación que realmente finaliza la orden.

97

```
void SetFinishRealStationType(byte Type)
```

Descripción general

Este método carga el tipo de estación que realmente finaliza la orden.

Retorno

```
void
```

Este método no retorna nada.

Parámetros

1. `byte Type`
   Tipo de estación que realmente finaliza la orden.

98

```
void SetFinishSide(byte Side)
```

Descripción general

Este método carga el lado en la que se finaliza la orden.

Retorno

```
void
```

Este método no retorna nada.

Parámetros

1. `byte Side`
   Lado donde realmente se finaliza la orden.

99

```
void SetFlagsIIS(dword Flags)
```

Descripción general

Este método prepara los datos de Flag de Palet en PIE. Éste es el flag que se transmitirá al Agente de Transportes cuando se notifique el palet en PIE.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `dword Flags`
   Valor del flag de palet en PIE.

100

```
void SetHeightType(byte HeightType)
```

Descripción general

Establece el valor del tipo de altura del tracking seleccionado.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `byte HeghtType`
   Valor del tipo de altura que se desea forzar.

101

```
void SetIISAux(byte Index,
               byte Value)
```

Descripción general

Este método prepara los datos auxiliares para el PIE que se van a notificar al Agente de Transportes.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `byte Index`
   Índice del dato que se va a enviar
2. `byte Value`
   Valor del dato que se desea almacenar

102

```
void SetIISData(byte Position,
                byte Value)
```

Descripción general

Este método prepara los datos de PIE que se van a notificar al Agente de Transportes.

Retorno

```
void
```
Este método no retorna nada.

Parámetros

1. `byte` `Position`
   Lugar en el cual se archivará el dato (valores comprendidos entre 0 y 255)
2. `byte` `Value`
   Valor que se desea almacenar

103

```
void SetIISHeightType(byte HeightType)
```

Descripción general

Este método prepara los datos del tipo de altura para el PIE que se van a notificar al Agente de Transportes.

Retorno

```
void
```
Este método no retorna nada.

Parámetros

1. `byte` `HeightType`
   Tipo de altura del contenedor de la orden que va a dar palet en PIE

104

```
void SetIISPlatformType(byte PlatformType)
```

Descripción general

Este método prepara los datos del tipo de plataforma para el PIE que se van a notificar al Agente de Transportes.

Retorno

```
void
```
Este método no retorna nada.

Parámetros

2. `byte` `PlataformType`
   Tipo de plataforma del contenedor de la orden que va a dar palet en PIE

105

```
void SetIISStationNumber(dword Number)
```

Descripción general

Este método prepara el número de la estación que a notificar Palet en PIE al Agente de Transportes.

Retorno

```
void
```
Este método no retorna nada.

Parámetros

1. `dword` `Number`
   Número de la estación que notificará el palet en PIE.

106

```
void SetIISStationType(dword Type)
```

Descripción general

Este método prepara el tipo de la estación que a notificar Palet en PIE al Agente de Transportes.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `dword Type`
   Tipo de la estación que notificará el palet en PIE.

107

```
void SetIISTransport(dword Transport)
```

Descripción general

Este método prepara los datos del numero de transporte para el PIE que se van a notificar al Agente de Transportes.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `dword Transport`
   Número de transporte de la orden que va a dar palet en PIE

108

```
void SetIISWeight(dword Weight)
```

Descripción general

Este método prepara los datos del peso para el PIE que se van a notificar al Agente de Transportes.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `dword Weight`
   Peso del contenedor de la orden que va a dar palet en PIE

109

```
void SetSelector(dword SelectNumber)
```

Descripción general

Este método realiza un cambio de selector en el grafo de la máquina. Un cambio de selector implica un reseteo a la etapa inicial del selector indicado. Como es normal en el flujo de programa dicho cambio realizará las correspondientes invocaciones a acciones de salida y entrada de las etapas en cuestión.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `dword SelectNumber`
   Número de selector al que se desea cambiar el grafo de funcionamiento.

110

```
void SetSourceAisle(byte SourceAisle)
```

Descripción general

Este método sirve para establecer el pasillo de origen del último tracking seleccionado.

Retorno

void

Este método no retorna nada.

Parámetros

1. byte SourceAisle
   Nuevo pasillo de origen

111

```
void SetSourceDepth(byte SourceDepth)
```

Descripción general

Este método sirve para establecer la profundidad de origen del último tracking que se haya seleccionado.

Retorno

void

Este método no retorna nada.

Parámetros

1. byte SourceDepth
   Nueva profundidad de origen

112

```
void SetSourceLocalX(byte X)
```

Descripción general

Este método carga la coordenada X (origen) en el tracking seleccionado.

Retorno

void

Este método no retorna nada.

Parámetros

1. byte X
   Coordenada X de origen.

113

```
void SetSourceLocalY(byte Y)
```

Descripción general

Este método carga la coordenada Y (origen) en el tracking seleccionado.

Retorno

void

Este método no retorna nada.

Parámetros

1. byte Y
Coordenada Y de origen.

114

```
void SetSourceNumber(byte SourceNumber)
```

Descripción general

Este método sirve para establecer el número de entidad origen del último tracking seleccionado.

Retorno

void
Este método no retorna nada.

Parámetros

1. byte SourceNumber
Nuevo número de entidad origen

115

```
void SetSourceSide(byte SourceSide)
```

Descripción general

Este método sirve para establecer el lado de origen para el último tracking seleccionado.

Retorno

void
Este método no retorna nada.

Parámetros

1. byte SourceSide
Nuevo lado de origen

116

```
void SetSourceType(byte SourceType)
```

Descripción general

Este método sirve para establecer el tipo de entidad origen del último tracking seleccionado.

Retorno

void
Este método no retorna nada.

Parámetros

1. byte SourceType
Nuevo tipo de entidad origen

117

```
void SetSourceX(word SourceX)
```

Descripción general

Este método sirve para establecer la coordenada X origen del último tracking seleccionado.

Retorno

void

Este método no retorna nada.

Parámetros

1. word SourceX

Nueva coordenada X de origen

118

```
void SetSourceY(word SourceX)
```

Descripción general

Este método sirve para establecer la coordenada Y origen del último tracking seleccionado.

Retorno

void

Este método no retorna nada.

Parámetros

1. word SourceY

Nueva coordenada Y de origen

119

```
void SetStationStatus(byte StationType,
                      byte StationNumber,
                      byte Status,
                      byte Loaded,
                      byte Capacity,
                      byte CurrentCount,
                      byte Aisle)
```

Descripción general

Este método actualiza la estación en la base de datos

Retorno

Este método no retorna nada.

Parámetros

1. byte StationType

Tipo de estación a actualizar.

2. byte StationNumber

Número de estación a actualizar.

3. byte Status

Estado actual de la estación, con los siguientes valores:

a. 0 → Defecto
b. 1 → En servicio

4. byte Loaded

Presencia en la estación que se va a actualizar:

a. 0 → Cargado
b. 1 → Vacío

5. byte Capacity

Capacidad actual de la estación.

6. byte CurrentCount

Ocupación actual de la estación.

7. byte Aisle

Pasillo actual de la estación a actualizar.

120

```
void SetTargetAisle(byte TargetAisle)
```

Descripción general

Este método sirve para establecer el pasillo destino para el último tracking que se haya seleccionado.

Retorno

```
void
```

Este método no devuelve nada

Parámetros

1. `byte TargetAisle`
   Nuevo pasillo destino de la orden

121

```
void SetTargetDepth(byte TargetDepth)
```

Descripción general

Este método sirve para establecer la profundidad destino para el último tracking que se haya seleccionado.

Retorno

```
void
```

Este método no devuelve nada

Parámetros

1. `byte TargetDepth`
   Nueva profundidad destino de la orden

122

```
void SetTargetLocalX(byte X)
```

Descripción general

Este método carga la coordenada X (destino) en el tracking seleccionado.

Retorno

```
void
```

Este método no retorna nada.

Parámetros

1. `byte X`
   Coordenada X de destino.

123

```
void SetTargetLocalY(byte Y)
```

Descripción general

Este método carga la coordenada Y (destino) en el tracking seleccionado.

Retorno

```
void
```

Este método no retorna nada.

Parámetros

1. byte Y
Coordenada Y de destino.

124

```
void SetTargetNumber(byte TargetNumber)
```

Descripción general

Este método sirve para establecer el número de destino del último tracking que se haya seleccionado.

Retorno

void
Este método no devuelve nada

Parámetros

1. byte TargetNumber
Nuevo número de entidad destino de la orden

125

```
void SetTargetSide(word TargetSide)
```

Descripción general

Este método sirve para establecer el lado de destino para el último tracking que se haya seleccionado.

Retorno

void
Este método no devuelve nada

Parámetros

1. word TargetSide
Nuevo lado de destino de la orden

126

```
void SetTargetType(byte TargetType)
```

Descripción general

Este método sirve para establecer el tipo de destino del último tracking que se haya seleccionado.

Retorno

void
Este método no devuelve nada

Parámetros

1. byte TargetType
Nuevo tipo de entidad destino de la orden

127

```
void SetTargetX(word TargetX)
```

Descripción general

Este método sirve para establecer la coordenada X de destino para el último tracking que se haya seleccionado.

Retorno

void

Este método no devuelve nada

Parámetros

1. word TargetX
   Nueva coordenada X de destino de la orden

128

```
void SetTargetY(word TargetY)
```

Descripción general

Este método sirve para establecer la coordenada Y de destino para el último tracking que se haya seleccionado.

Retorno

void
Este método no devuelve nada

Parámetros

1. word TargetY
   Nueva coordenada Y de destino de la orden

CONCEPTOS GENERALES DE TEMPORIZADORES

A partir de la versión 3.0 del Sistema de Control Galileo se incorpora el concepto de Temporizador por máquina. Todas las máquinas pueden gestionar 255 temporizadores numerados de 0 al 254 para todas sus operaciones internas. Estos temporizadores pueden ser de cualquier tipo y el tipo de cada temporizador se define en la utilización.

129

```
void SetTimerSA(dword TempNumber,
                dword Tempmseg)
```

Descripción general

Configura el temporizador indicado como RETARDO A LA DESCONEXIÓN

Retorno

void
Este método no retorna nada

Parámetros

1. dword TempNumber
   Número de temporizador que se desea utilizar, dentro del rango de 0 a 254.
2. dword Tempmseg

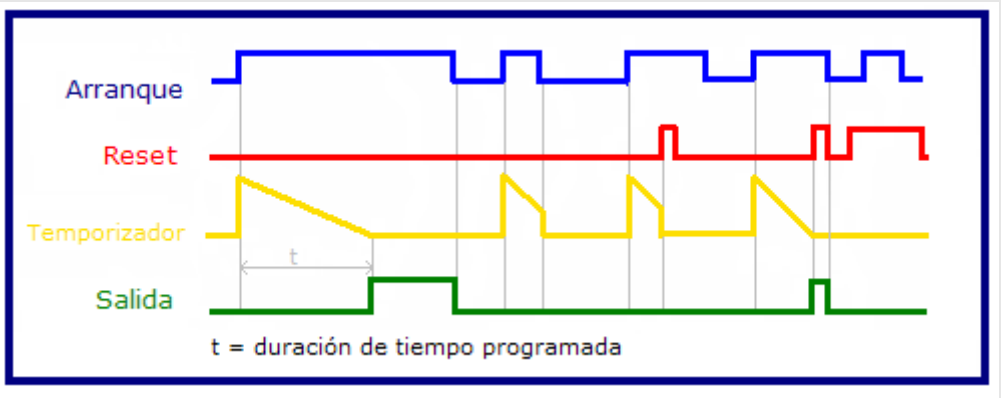
Tiempo en mseg. de la temporización.

130

```
void SetTimerSE(dword TempNumber,
                dword Tempmseg)
```

Descripción general

Configura el temporizador indicado como RETARDO A LA CONEXIÓN



Retorno

`void`

Este método no retorna nada

Parámetros

1. `dword TempNumber`

   Número de temporizador que se desea utilizar, dentro del rango de 0 a 254.

2. `dword Tempmseg`

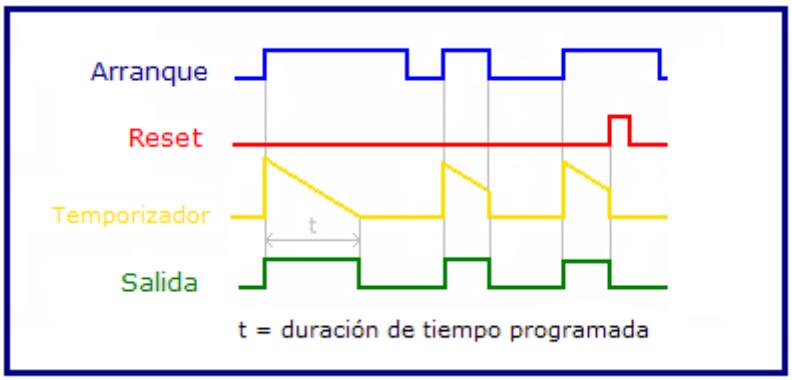
   Tiempo en mseg. de la temporización.

131

```
void SetTimerSI(dword TempNumber,
                dword Tempmseg)
```

Descripción general

Configura el temporizador indicado como TEMPORIZADOR DE IMPULSO.



Retorno

`void`

Este método no retorna nada

Parámetros

1. `dword` `TempNumber`

   Número de temporizador que se desea utilizar, dentro del rango de 0 a 254.

2. `dword` `Tempmseg`

   Tiempo en mseg. de la temporización.

132

```
void SetTimerSS(dword TempNumber,
                dword Tempmseg)
```

Descripción general

Configura el temporizador indicado como RETARDO A LA CONEXIÓN CON MEMORIA

Retorno

`void`

Este método no retorna nada

Parámetros

1. `dword` `TempNumber`

   Número de temporizador que se desea utilizar, dentro del rango de 0 a 254.

2. `dword` `Tempmseg`

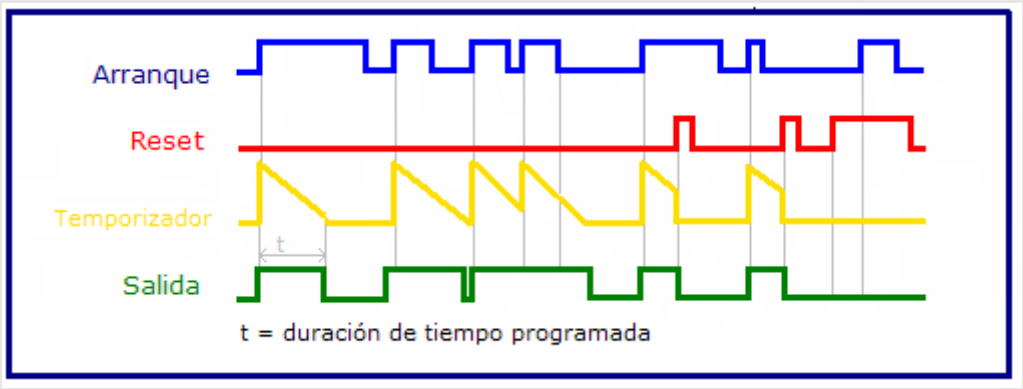
   Tiempo en mseg. de la temporización.

133

```
void SetTimerSV(dword TempNumber,
                dword Tempmseg)
```

Descripción general

Configura el temporizador indicado como TEMPORIZADOR CON IMPULSO PROLONGADO.



Retorno

`void`

Este método no retorna nada

Parámetros

1. `dword TempNumber`
   Número de temporizador que se desea utilizar, dentro del rango de 0 a 254.
2. `dword Tempmseg`
   Tiempo en mseg. de la temporización.

134

```
void SetTrackingData(byte Position,
                     byte Value)
```

Descripción general

La estructura del tracking posee un área de datos utilizables por el usuario. Este método permite escribir en esta área. Hay que hacer notar que estos datos son arrastrados con el tracking.

Retorno

`void`

Este método no devuelve nada

Parámetros

1. `byte Position`
   Posición del área de datos en la cual se desea escribir. Los valores posibles van desde 0 a 49.
2. `byte Value`
   Valor que se desea almacenar en la posición indicada.

135

```
void SetTransportNumber(dword Number)
```

Descripción general

Este método sirve para establecer el número de transporte del último tracking seleccionado. Nótese que si este número se establece a 0, este tracking ya no contará para la cuenta de trackings de la máquina.

Retorno

void

Este método no retorna nada.

Parámetros

1. dword Number

   Número de transporte nuevo

136

```
void SetVarValue(string Variable,
                 dword Value)
```

Descripción general

Este método realiza un cambio del valor de una variable de forma indirecta.

Retorno

void

Este método no retorna ningún valor.

Parámetros

1. string Variable

   Nombre de la variable sobre la que se desea actual. Debe de ser una variable de dicha máquina.

2. dword Value

   Nuevo valor que se desea proporcionar a la variable indicada en el parámetro anterior.

137

```
void SetWTUNumber(dword WTUNumber)
```

Descripción general

Este método sirve para establecer el número de WTU (UMA) del último tracking seleccionado.

Retorno

void

Este método no retorna nada.

Parámetros

1. dword WTUNumber

   WTU nuevo

138

```
void SetWTUType(byte WTUType)
```

Descripción general

Este método sirve para establecer el tipo de WTU (UMA) del último tracking que se haya seleccionado.

Retorno

```
void
```
Este método no retorna nada.

Parámetros

1. `byte WTUType`
   Nuevo tipo de entidad WTU

139

```
void SimulateSignal(bool &SimulateSignal,
                    bool StartCond,
                    bool StopCond,
                    dword ActivateTime,
                    dword DesactivateTime)
```

Descripción general

Este método se utiliza típicamente en las funciones de simulación. Su principal utilidad consiste en permitir la activación desactivación de señales en función de ciertas condiciones mantenidas con el tiempo.

Retorno

```
void
```
Este método no retorna nada

Parámetros

1. `bool SimulateSignal`
   Señal que se desea simular. Esta señal es pasada por referencia y por lo tanto debe de ser una variable definida en el simbólico de variables del software de desarrollo Designer. No es admisible una variable local de la función.
2. `bool StartCond`
   Esta es la condición para poder contar el tiempo de activación. La señal simulada se activará cuando esta condición este activa durante el tiempo de acti vación.
3. `bool StopCond`
   Esta es la condición para poder contar el tiempo de desactivación. La señal simulada se desactivará cuando esta condición esté activa durante el tiempo de desactivación.
4. `dword ActivateTime`
   Tiempo acumulado que debe de estar activa la condición de activación para poder activar la señal simulada. No es necesario que dicha condición esté activa continuamente, ya que el tiempo se acumula.
5. `dword DesactivateTime`
   Tiempo acumulado de desactivación para desactivar la señal simulada. No es necesario que dicho condición esté activa continuamente, ya que el tiempo se acumula.

140

```
void TerminateCommand()
```

Descripción general

Este método informa al Agente de Transportes que se desea finalizar la orden que ha sido antes preparada para finalizar.

Retorno

```
void
```
Este método no retorna nada.

Parámetros

Este método no tiene parámetros de entrada

141

```
dword TrackingCount()
```

Descripción general

Este método retorna la cantidad de trackings con datos que tiene la máquina.

Retorno

dword

Retorna la cantidad de trackings con datos que tiene la máquina.

Parámetros

Este método no tienen parámetros de entrada

142

```
word GetPriority()
```

Descripción general

Este método retorna la prioridad asociada al tracking actual seleccionado.

Retorno

word

Retorna el campo "prioridad" del tracking seleccionado.

Parámetros

Este método no tienen parámetros de entrada

143

```
word GetSequence()
```

Descripción general

Este método retorna el número de secuencia asociado al tracking actual seleccionado.

Retorno

word

Retorna el campo "secuencia" del tracking seleccionado.

Parámetros

Este método no tienen parámetros de entrada

144

```
word GetSpeedX()
```

Descripción general

Este método retorna la velocidad/aceleración en el eje X asociada al tracking actual seleccionado.

Retorno

word
Retorna el campo "VelocidadX" del tracking seleccionado.

Parámetros

Este método no tienen parámetros de entrada

145

```
word GetSpeedY()
```

Descripción general

Este método retorna la velocidad/aceleración en el eje Y asociada al tracking actual seleccionado.

Retorno

word
Retorna el campo "VelocidadY" del tracking seleccionado.

Parámetros

Este método no tienen parámetros de entrada

146

```
word GetSpeedZ()
```

Descripción general

Este método retorna la velocidad/aceleración en el eje Z asociada al tracking actual seleccionado.

Retorno

word
Retorna el campo "VelocidadZ" del tracking seleccionado.

Parámetros

Este método no tienen parámetros de entrada

147

```
void SetPriority( word aValue )
```

Descripción general

Este método establece la prioridad asociada al tracking actual seleccionado.

Retorno

Este método no retorna nada

Parámetros

1. word aValue
   Valor de prioridad a establecer

148

```
void SetSequence(word aValue)
```

Descripción general

Este método establece el número de secuencia asociado al tracking actual

seleccionado.

Retorno

Este método no retorna nada

Parámetros

1. word aValue
   Valor de secuencia a establecer

149

```
void SetSpeedX(word aValue)
```

Descripción general

Este método establece la velocidad/aceleración en el eje X asociada al tracking actual seleccionado.

Retorno

Este método no retorna nada

Parámetros

1. word aValue
   Valor de velocidad a establecer

150

```
void SetSpeedY(word aValue)
```

Descripción general

Este método establece la velocidad/aceleración en el eje Y asociada al tracking actual seleccionado.

Retorno

Este método no retorna nada

Parámetros

1. word aValue
   Valor de velocidad a establecer

151

```
void SetSpeedZ(word aValue)
```

Descripción general

Este método establece la velocidad/aceleración en el eje Z asociada al tracking actual seleccionado.

Retorno

Este método no retorna nada

Parámetros

1. word aValue
   Valor de velocidad a establecer

152

```
void SetIISLocalX(word aValue)
```

Descripción general

Este método establece la coordenada X local asociada al evento

Retorno

Este método no retorna nada

Parámetros

1. word aValue
   Valor de la coordenada a establecer

153

```
void SetIISLocalY(word aValue)
```

Descripción general

Este método establece la coordenada Y local asociada al evento

Retorno

Este método no retorna nada

Parámetros

1. word aValue
   Valor de la coordenada a establecer

154

```
void SetIISDepth(byte aValue)
```

Descripción general

Este método establece la profundidad asociada al evento

Retorno

Este método no retorna nada

Parámetros

1. byte aValue
   Valor de la profundidad a establecer

155

```
void SetIISAisle(byte aValue)
```

Descripción general

Este método establece el pasillo asociada al evento

Retorno

Este método no retorna nada

Parámetros

1. byte aValue
   Valor del pasillo a establecer

156

```
void SetMultiOPIISFlags(byte slot, dword value)
```

Descripción general

Este método establece el valor de los flags del evento, en el slot seleccionado

Retorno

Este método no retorna nada

Parámetros

1. byte slot
   Slot en el que se establecen los datos
2. dword value
   Valor de flags a establecer

157

```
void SetMultiOPIISStationNumber(byte slot, byte value)
```

Descripción general

Este método establece el numero de estación asociada del evento, en el slot seleccionado

Retorno

Este método no retorna nada

Parámetros

1. byte slot
   Slot en el que se establecen los datos
2. byte value
   Número de estación a establecer

158

```
void SetMultiOPIISStationType(byte slot, byte value)
```

Descripción general

Este método establece el tipo de estación asociada del evento, en el slot seleccionado

Retorno

Este método no retorna nada

Parámetros

1. byte slot
   Slot en el que se establecen los datos
2. byte value
   Tipo de estación a establecer

159

```
void SetMultiOPIISAuxData(byte slot, byte index, byte value)
```

Descripción general

Este método establece un byte en la zona de datos auxiliares del evento, en el slot seleccionado.

Retorno

Este método no retorna nada

Parámetros

1. byte slot
Slot en el que se establecen los datos
2. byte index
Índice del byte a establecer
3. byte value
Valor a establecer

160

```
byte GetMultiOPIISAuxData(byte slot, byte index)
```

Descripción general

Este método devuelve un byte de la zona de datos auxiliares del evento, en el slot seleccionado.

Retorno

El valor del byte pedido

Parámetros

1. byte slot
Slot en el que se buscan los datos
2. byte index
Índice del byte a devolver

161

```
void SetMultiOPIISTransport(byte slot, dword transport)
```

Descripción general

Este método establece el numero de transporte asociado del evento, en el slot seleccionado

Retorno

Este método no retorna nada

Parámetros

1. byte slot
Slot en el que se establecen los datos
2. dword transport
Número de transporte a establecer

162

```
void SetMultiOPIISWeight(byte slot, dword weight)
```

Descripción general

Este método establece el peso asociado del evento, en el slot seleccionado

Retorno

Este método no retorna nada

Parámetros

1. byte slot
Slot en el que se establecen los datos
2. dword weight
Peso a establecer

163

```
void SetMultiOPIISPlatformType(byte slot, byte platformType)
```

Descripción general

Este método establece el tipo de contenedor asociado del evento, en el slot seleccionado

Retorno

Este método no retorna nada

Parámetros

1. byte slot
   Slot en el que se establecen los datos
2. byte platformType
   Tipo de contenedor a establecer

164

```
void SetMultiOPIISHeightType(byte slot, byte heightType)
```

Descripción general

Este método establece el tipo de altura asociado del evento, en el slot seleccionado

Retorno

Este método no retorna nada

Parámetros

1. byte slot
   Slot en el que se establecen los datos
2. byte heightType
   Tipo de altura a establecer

165

```
void SetMultiOPIISX(byte slot, word x)
```

Descripción general

Este método establece la coordenada X del evento, en el slot seleccionado

Retorno

Este método no retorna nada

Parámetros

1. byte slot
   Slot en el que se establecen los datos
2. word x
   Coordenada X a establecer

166

```
void SetMultiOPIISY(byte slot, word y)
```

Descripción general

Este método establece la coordenada Y del evento, en el slot seleccionado

Retorno

Este método no retorna nada

Parámetros

1. byte slot
Slot en el que se establecen los datos
2. word y
Coordenada Y a establecer

167

```
void SetMultiOPIISSide(byte slot, byte side)
```

Descripción general

Este método establece el lado del evento, en el slot seleccionado

Retorno

Este método no retorna nada

Parámetros

1. byte slot
Slot en el que se establecen los datos
2. byte side
Lado a establecer

168

```
void SetMultiOPIISEventType(byte slot, byte eventType)
```

Descripción general

Este método establece el tipo de evento, en el slot seleccionado

Retorno

Este método no retorna nada

Parámetros

1. byte slot
Slot en el que se establecen los datos
2. byte eventType
Tipo de evento a establecer

169

```
void SetMultiOPIISLocalX(byte slot, byte localX)
```

Descripción general

Este método establece la coordenanda X local del evento, en el slot seleccionado

Retorno

Este método no retorna nada

Parámetros

1. byte slot
Slot en el que se establecen los datos
2. byte localX
Coordenada X a establecer

170

```
void SetMultiOPIISLocalY(byte slot, byte localY)
```

Descripción general

Este método establece la coordenanda Y local del evento, en el slot seleccionado

Retorno

Este método no retorna nada

Parámetros

1. byte slot
   Slot en el que se establecen los datos
2. byte localY
   Coordenada Y a establecer

171

```
void SetMultiOPIISDepth(byte slot, byte depth)
```

Descripción general

Este método establece la profundidad del evento, en el slot seleccionado

Retorno

Este método no retorna nada

Parámetros

1. byte slot
   Slot en el que se establecen los datos
2. byte depth
   Profundidad a establecer

172

```
void SetMultiOPIISAisle(byte slot, byte aisle)
```

Descripción general

Este método establece el pasillo del evento, en el slot seleccionado

Retorno

Este método no retorna nada

Parámetros

1. byte slot
   Slot en el que se establecen los datos
2. byte aisle
   Pasillo a establecer

173

```
void SetMultiOPIISAux(byte slot, byte index, byte value)
```

Descripción general

Este método establece un valor en los datos auxiliares del evento, en el slot seleccionado

Retorno

Este método no retorna nada

Parámetros

1. byte slot
   Slot en el que se establecen los datos
2. byte index
   Indice del byte a establecer

3. byte value
Valor del byte a establecer

174

```
void ClearMultiOPIISData(byte slot)
```

Descripción general

Este método borra la zona de datos auxiliares del evento, en el slot indicado

Retorno

Este método no retorna nada

Parámetros

1. byte slot
Slot en el que se establecen los datos

175

```
void SetMultiOPEndRealStationNumber(byte slot, byte number)
```

Descripción general

Este método establece el número de estación de la zona de finalización en el slot indicado.

Retorno

Este método no retorna nada

Parámetros

1. byte slot
Slot en el que se establecen los datos.
2. byte number
Número de estación a establecer.

176

```
void SetMultiOPEndRealStationType(byte slot, byte type)
```

Descripción general

Este método establece el tipo de estación de la zona de finalización en el slot indicado.

Retorno

Este método no retorna nada

Parámetros

1. byte slot
Slot en el que se establecen los datos.
2. byte type
Tipo de estación a establecer

177

```
void SetMultiOPEndPlatformType(byte slot, byte platformType)
```

Descripción general

Este método establece el tipo de plataforma de la zona de finalización en el slot indicado.

Retorno

Este método no retorna nada

Parámetros

1. `byte slot`
   Slot en el que se establecen los datos.
2. `byte platformType`
   Tipo de plataforma a establecer

178

```
void SetMultiOPEndHeightType(byte slot, byte heightType)
```

Descripción general

Este método establece el tipo de altura finalización en el slot indicado.

Retorno

Este método no retorna nada

Parámetros

1. `byte slot`
   Slot en el que se establecen los datos.
2. `byte heightType`
   Tipo de altura a establecer

179

```
void SetMultiOPEndLocalX(byte slot, byte localX)
```

Descripción general

Este método establece la X local de finalización en el slot indicado.

Retorno

Este método no retorna nada

Parámetros

1. `byte slot`
   Slot en el que se establecen los datos.
2. `byte localX`
   Valor de X local a establecer.

180

```
void SetMultiOPEndLocalY(byte slot, byte localY)
```

Descripción general

Este método establece la Y local de finalización en el slot indicado.

Retorno

Este método no retorna nada

Parámetros

1. `byte slot`
   Slot en el que se establecen los datos.

2. byte localY
Valor de Y local a establecer.

181

```
void SetMultiOPEndSide(byte slot, byte side)
```

Descripción general

Este método establece el lado de finalización en el slot indicado.

Retorno

Este método no retorna nada

Parámetros

1. byte slot
Slot en el que se establecen los datos.
2. byte side
Lado a establecer.

182

```
void SetMultiOPEndAux(byte slot, byte index, byte value)
```

Descripción general

Este método establece un valor en los datos auxiliares de la zona de finalización.

Retorno

Este método no retorna nada

Parámetros

1. byte slot
Slot en el que se establecen los datos.
2. byte index
Índice del byte a establecer.
3. byte value
Valor del byte a establecer.

183

```
void SetMultiOPEndLogicalX(byte slot, byte logicalX)
```

Descripción general

Este método establece la X lógica de finalización.

Retorno

Este método no retorna nada

Parámetros

1. byte slot
Slot en el que se establecen los datos.
2. byte logicalX
Valor de X lógica a establecer.

184

```
void SetMultiOPEndLogicalY(byte slot, byte logicalY)
```

Descripción general

Este método establece la Y lógica de finalización.

Retorno

Este método no retorna nada

Parámetros

1. byte slot
   Slot en el que se establecen los datos.
2. byte logicalY
   Valor de Y lógica a establecer.

185

```
void SetMultiOPEndDepth(byte slot, byte depth)
```

Descripción general

Este método establece la profundidad de finalización.

Retorno

Este método no retorna nada

Parámetros

1. byte slot
   Slot en el que se establecen los datos.
2. byte logicalY
   Valor de la profundidad a establecer.

186

```
void SetMultiOPEndAisle(byte slot, byte aisle)
```

Descripción general

Este método establece el pasillo de finalización.

Retorno

Este método no retorna nada

Parámetros

1. byte slot
   Slot en el que se establecen los datos.
2. byte aisle
   Pasillo a establecer.

187

```
void PrepapeMultiOPCommandToFinish(byte slot, byte tracking, dword
errorCode, dword aux)
```

Descripción general

Este método prepara la multioperación a realizar (para la finalización de order).

Retorno

Este método no retorna nada

Parámetros

1. `byte slot`
   Slot en el que se almacenará.
2. `byte tracking`
   Tracking a utilizar.
3. `dword errorCode`
   Código de error.
4. `dword aux`
   Datos auxiliares a utilizar.

188

```
void SetMultiOPSearchCurrentX(word X)
```

Descripción general

Este método establece la X actual de búsqueda.

Retorno

Este método no retorna nada

Parámetros

1. `word X`
   Valor de la X actual a establecer.

189

```
void SetMultiOPSearchCurrentY(word Y)
```

Descripción general

Este método establece la Y actual de búsqueda.

Retorno

Este método no retorna nada

Parámetros

1. `word Y`
   Valor de la Y actual a establecer.

190

```
void SetMultiOPSearchCurrentSide(byte side)
```

Descripción general

Este método establece el lado de búsqueda.

Retorno

Este método no retorna nada

Parámetros

1. `byte side`
   Valor del lado a establecer.

191

```
void SetMultiOPSearchCurrentAisle(byte aisle)
```

Descripción general

Este método establece el pasillo de búsqueda.

Retorno

Este método no retorna nada

Parámetros

1. byte aisle
   Valor del pasillo a establecer.

192

```
void SetMultiOPSearchCurrentTransport(byte transport)
```

Descripción general

Este método establece el número de transporte de búsqueda.

Retorno

Este método no retorna nada

Parámetros

1. byte aisle
   Valor del transorte a establecer.

193

```
void SetMultiOPSearchCurrentCapacity(byte capacity)
```

Descripción general

Este método establece el valor de la capacidad de búsqueda.

Retorno

Este método no retorna nada

Parámetros

1. byte capacity
   Valor de la capacidad a establecer.

194

```
void SetMultiOPSearchCurrentOcupation(byte occupation)
```

Descripción general

Este método establece el valor de la ocupación de búsqueda.

Retorno

Este método no retorna nada

Parámetros

1. byte occupation
   Valor de la ocupación a establecer.

195

```
void SetMultiOPSearchStationNumber(byte stationNumber)
```

Descripción general

Este método establece el número de estación de búsqueda.

Retorno

Este método no retorna nada

Parámetros

1. byte stationNumber
   Valor del número de estación a establecer.

196

```
void SetMultiOPSearchStationType(byte stationType)
```

Descripción general

Este método establece el tipo de estación de búsqueda.

Retorno

Este método no retorna nada

Parámetros

1. byte stationType
   Valor del tipo de estación a establecer.

197

```
void SetMultiOPSearchAuxData(byte index, byte value)
```

Descripción general

Este método establece un valor auxiliar de la zona de búsqueda.

Retorno

Este método no retorna nada

Parámetros

1. byte index
   Índice del valor a establecer.
2. byte value
   Valor a establecer.

198

```
void ExecuteMultiOP(bool searchActive, byte endCounter, byte
issCounter)
```

Descripción general

Este método ejecuta la multioperación.

Retorno

Este método no retorna nada

Parámetros

1. bool searchActive
Índica si se desea realizar una búsqueda.
2. byte endCounter
Número de operaciones de finalización.
3. byte issCounter
Número de operaciones de evento.

199

```
bool IsMultiOPFinished(void)
```

Descripción general

Este método indica si se ha completado la multioperación.

Retorno

True si la multioperación finalizó, false en caso contrario.

Parámetros

Este método no tiene parámetros.

200

```
void ExitMultiOP(void)
```

Descripción general

Este método finaliza la multioperación.

Retorno

Este método no retorna nada

Parámetros

Este metodo no tiene parametros.

4.1.4.2 Categorías de Traza

Las siguientes son las categorías registradas de traza para dicho componente:

```
# Componente de gestión de Maquina (Machine )
# ============================
log4j.category.Galileo.Components.Machine.CopyTracking=ERROR, A2
Traza las copias de tracking desde código XANA en modo DEBUG
log4j.category.Galileo.Components.Machine.InterchangeTracking=ERROR, A2
Traza los intercambios de tracking desde código XANA en modo INFO o superior.
log4j.category.Galileo.Components.Machine.ClearTracking=ERROR, A2
Traza las eliminaciones de un tracking desde código XANA en modo INFO o superior.
log4j.category.Galileo.Components.Machine.ClearAllTrackings=ERROR, A2
Traza las eliminaciones de todos los tracking de una máquina desde código XANA en modo INFO o
superior.
log4j.category.Galileo.Components.Machine.GetTracking=ERROR, A2
Traza las adquisiciones de tracking desde código XANA en modo INFO o superior.
log4j.category.Galileo.Components.Machine.SetTimer=ERROR, A2
Traza la inicialización de temporizadores en modo DEBUG o superior.
log4j.category.Galileo.Components.Machine.EvalTimer=ERROR, A2
Traza la Evaluación de temporizadores en modo DEBUG o superior.
log4j.category.Galileo.Components.Machine.ResetTimer=ERROR, A2
Traza el reseteo de temporizadores en modo DEBUG o superior.
log4j.category.Galileo.Components.Machine.RestartTimer=ERROR, A2
Traza el reinicio de la cuenta de temporizadores en modo DEBUG o superior.
log4j.category.Galileo.Components.Machine.GetTimerCount=ERROR, A2
Traza la consulta sobre la cuenta actual de temporizadores en modo DEBUG o superior.
log4j.category.Galileo.Components.Machine.Simulation=ERROR, A2
Traza la simulación de señales en modo DEBUG o superior.
log4j.category.Galileo.Components.Machine.TrackingCount=ERROR, A2
Traza los transportes activos de una máquina si se pone en modo DEBUG o superior.
log4j.category.Galileo.Components.Machine.Reset=INFO, A2
Traza el paso al que se fuerza una maquina si se pone en modo DEBUG o superior.
log4j.category.Galileo.Components.Machine.SetTrackingData=ERROR, A2
Traza los datos con los que se sobreescribe el tracking de una maquina en una intervencion manual si se
pone en modo DEBUG o superior.
log4j.category.Galileo.Components.Machine.SetTimer=ERROR, A2
Traza el establecimiento de un nuevo Timer cuando se pone en modo DEBUG o superior.
log4j.category.Galileo.Components.Machine.HoldTimeout=ERROR,A2
```

Traza los ciclos que no se cuentan para timeout en las etapas en que se active este método cuando se pone en modo DEBUG.

4.1.5 POSITIONMAP

4.1.5.1 Interface

Este componente es básicamente un archivo de configuración modificado para soportar algún método más y alguna característica extra.

La primera característica es que cada posición viene definida por una entrada en este archivo, la entrada con número 0 debe existir siempre y no tiene un valor posicional, sino que se trata del Offset (valor que se sumará a todas las posiciones cuando se consulten.

Como ejemplo podemos tomar este archivo de posiciones en el eje de elevación Y:

```
0= 40000  // Offset (se suma a todas las demás)
1= 683    // Posición 1 de elevación en mm.
2= 5913   // Posición 2 de elevación en mm.
3= 11169  // Posicion 3 de elevación en mm.
4= 16424  // Posición 4 de elevación en mm.
```

```
5= 21679  // Posición 5 de elevación en mm.
```

El interface correspondiente es:

Class	PositionMap
BUS IN	0 bytes
BUS OUT	0 bytes

Listado de métodos:

1	bool Add(dword Parameter, dword Value)
2	void Del(dword Parameter)
3	bool Exists(dword Parameter)
4	dword GetLowerPosition(dword Position, dword Delta)
5	dword GetNearestPosition(dword Position)
6	dword GetUpperPosition(dword Position, dword Delta)
7	bool Load(string File)
8	bool LoadAndTrack(string File)
9	bool Modify(dword Parameter, dword Value)
10	dword Position(dword Position)
11	bool Sync()

Descripción de métodos:

1

```
bool Add(dword Parameter,
         dword Value)
```

Descripción general

Este método añade o sobrescribe un parámetro en memoria.

Retorno

`true`
Si la operación se ha realizado con éxito.
`false`
Si no se ha cargado ningún archivo previamente a la invocación.

Parámetros

1. `dword Parameter`
   Número de parámetro que se desea añadir
2. `dword Value`
   Valor que se desea asignar al parámetro.

2

```
void Del(dword Parameter)
```

Descripción general

Este método elimina un parámetro determinado.

Retorno

Este método no retorna nada.

Parámetros

1. `dword` `Parameter`
   Número de parámetro que se desea eliminar.

3

```
bool Exists(dword Parameter)
```

Descripción general

Este método consulta la existencia de un parámetro en el archivo de configuración.

Retorno

`true`
Si el parámetro indicado en `Parameter` existe en el archivo.
`false`
Si el parámetro indicado en `Parameter` NO existe en el archivo.

Parámetros

1. `dword` `Parameter`
   Número de parámetro que se desea consultar.

4

```
dword GetLowerPosition(dword Position,
                       dword Delta)
```

Descripción general

Este método permite averiguar cual es la siguiente posición lógica en el archivo de posiciones con el valor más próximo por debajo al valor físico indicado.

Retorno

`dword`
Devuelve la posición lógica inmediatamente inferior a la posición física indicada en el parámetro `Position`.

Parámetros

1. `dword` `Position`
   Posición física (cota) que se desea evaluar.
2. `dword` `Delta`
   Intervalo de margen para realizar la evaluación.

5

```
dword GetNearestPosition(dword Position)
```

Descripción general

Este método sirve para averiguar la posición lógica más cercana a la posición física actual.

Retorno

dword
Devuelve la posición lógica más cercana a la posición física indicada en el parámetro `Position`.

Parámetros

1. `dword` `Position`
   Posición física (cota) que se desea evaluar.

6

```
dword GetUpperPosition(dword Position,
                       dword Delta)
```

Descripción general

Este método permite averiguar cual es la siguiente posición lógica en el archivo de posiciones con el valor más próximo por arriba al valor físico indicado.

Retorno

dword
Devuelve la posición lógica inmediatamente superior a la posición física indicada en el parámetro `Position`.

Parámetros

1. `dword` `Position`
   Posición física (cota) que se desea evaluar.
2. `dword` `Delta`
   Intervalo de margen para realizar la evaluación.

7

```
bool Load(string File)
```

Descripción general

Carga en memoria el archivo especificado. Este método debe ser invocado en las rutinas de arranque ya que en el código cíclico supondría una carga de tiempo de ciclo el acceso al sistema de archivos.

Retorno

`true`
Si la carga del fichero ha tenido éxito.
`false`
Si la carga del fichero no se ha podido finalizar correctamente.

Parámetros

1. `string` `File`
   Nombre del archivo que se desea cargar (con ruta).

8

```
bool LoadAndTrack(string File)
```

Descripción general

Carga en memoria el archivo especificado. Este método debe ser invocado en las rutinas de arranque ya que en el código cíclico supondría una carga de tiempo de ciclo el acceso al sistema de archivos. La diferencia de este método con el otro método de carga estriba en que una vez cargado, el componente permanece pendiente de cualquier modificación que sufra el archivo de carga, y si se produce alguna, procede a recargar los datos del mismo. Esto evita tener que parar y

rearrancar Galileo para recargar una configuración.

Retorno

`true`
Si la operación se ha realizado con éxito.
`false`
La operación NO se ha realizado con éxito, es decir, el archivo no ha sido cargado.

Parámetros

1. `string` File
   Nombre del archivo que se desea cargar (especificando la ruta).

9

```
bool Modify(dword Parameter,
            dword Value)
```

Descripción general

Este método modifica un parámetro en memoria, o lo crea si no existe en memoria.

Retorno

`true`
Si la operación se ha realizado con éxito.
`false`
Si no se ha cargado ningún archivo previamente a la invocación.

Parámetros

1. `dword` Parameter
   Número de parámetro que se desea añadir
2. `dword` Value
   Valor que se desea asignar al parámetro.

10

```
dword Position(dword Position)
```

Descripción general

Retorna el valor de posición configurado para una coordenada dada. El valor retornado lleva sumado el offset.

Retorno

`dword`
Retorna el valor configurado más el offset.

Parámetros

1. `dword` Position
   Número de posición dentro del archivo que se desea consultar.

11

```
bool Sync()
```

Descripción general

Este método archiva en disco todos los cambios que se han producido en el archivo de configuración.

Retorno

`true`
Si la operación se ha realizado con éxito.

false Si no se ha cargado ningún archivo previamente
Parámetros
Este método no tiene parámetros de entrada.

4.1.6 SYSTEM

4.1.6.1 Interface

El objeto System encapsula cierta funcionalidad de acceso al sistema. Esta funcionalidad podrá ser ampliada en el futuro.

NOTA: A partir de la versión 3 de GALILEO existe una variable global de nombre g_System que permite el acceso a este componente sin declarar explícitamente una variable de este tipo.

Este componente expone el siguiente interface:

Class	System
BUS IN	0 bytes
BUS OUT	0 bytes

Listado de métodos:

1	dword ASCIIToNum(dword Number)
2	void EvalFailure(dword Failure, dword Machine, bool Signal, bool &Output)
3	bool ExistMachineByName(string MachineName)
4	bool ExistMachineByNumber(dword MachineNumber)
5	bool GetFlipFlopBit(dword Bit)
6	Machine GetMachineByName(string MachineName)
7	Machine GetMachineByNumber(dword MachineNumber)
8	bool IsDigit(dword Number)
9	bool IsErrorActive(dword ErrorNo)
10	bool IsLower(dword Number)
11	bool IsSpace(dword Number)
12	bool IsUpper(dword Number)
13	bool NegativeSignalChange(bool &Signal)
14	dword NumToASCII(dword Number)
15	bool PositiveSignalChange(bool &Signal)
16	void ResetAllFailures()
17	void ResetFailure(dword Failure, dword Machine)
18	void ResetGlobalFailures()

19	`void ResetMachineFailure(dword Machine)`
20	`void SetCancelEventResponse(dword TransportNumber, word Status)`
21	`void SetFailure(dword Failure, dword Machine)`
22	`void SetRouteStatus(byte SourceStationType, byte SourceStationNumber, byte TargetStationType, byte TargetStationNumber, byte Status)`
23	`bool SignalChange(bool &Signal)`
24	`void TransportCancelEventFinished(dword Transport)`
25	`bool TransportHasCancelEvent(dword Transport)`
26	`void TriggerWatchDog()`
27	`void WriteLog(dword d1, dword d2, dword d3, dword d4)`

Descripción de métodos:

1

```
dword ASCIIToNum(dword Number)
```

Descripción general

Este método realiza la conversión de un carácter ASCII correspondiente a un dígito a su correspondiente valor numérico.

Retorno

`dword`

Representación numérica que corresponde a dicho dígito.

Parámetros

1. `dword` Number
   Valor del número que se desea convertir a ASCII. Debe de tener un valor entre 0 y 9

2

```
void EvalFailure(dword Failure,
                 dword Machine,
                 bool Signal,
                 bool &Output)
```

Descripción general

Este método evalúa la posibilidad de la existencia de una avería.
Gestiona internamente la señalización a la visualización. Este es el método recomendado para la gestión de averías.

Retorno

`void`
Este método no retorna nada

Parámetros

1. `dword` `Failure`
   Número de fallo a evaluar
2. `dword` `Machine`
   Número de máquina que es autora del defecto, debe de ser 0 si es un defecto general.
3. `bool` `Signal`
   Señal booleana que genera la avería. Esta señal se interpreta con lógica negativa, es decir, si `Signal` es `true` no existe fallo alguno.
4. `bool` `Output`
   Es una variable booleana por referencia. La variable se modifica para señalar avería si se da la condición de avería, en cuyo caso pasará a tener el valor `true`. Si no hay condición de avería la variable no modifica su valor.

3

```
bool ExistMachineByName(string MachineName)
```

Descripción general

Este método sirve para averiguar si una máquina determinada, consultada por nombre de máquina existe. Es la comprobación previa a realizar antes de invocar cualquier metodo del estilo `GetMachineBy`…

Retorno

`true`
Existe la máquina con el nombre de máquina indicado en el parámetro `MachineName`
`false`
No existe la máquina con el nombre de máquina indicado en el parámetro `MachineName`

Parámetros

1. `string` `MachineName`
   Nombre de la máquina que se desea saber si existe.

4

```
bool ExistMachineByNumber(dword MachineNumber)
```

Descripción general

Este método sirve para averiguar si una máquina determinada, consultada por número de máquina existe. Es la comprobación previa a realizar antes de invocar cualquier metodo del estilo `GetMachineBy`…

Retorno

`true`
Existe la máquina con el número de máquina indicado en el parámetro `MachineNumber`
`false`
No existe la máquina con el número de máquina indicado en el parámetro `MachineNumber`

Parámetros

2. `dword MachineNumber`
Número de la máquina que se desea saber si existe.

5

```
bool GetFlipFlopBit(dword Bit)
```

Descripción general

Este método sirve para obtener uno de los bits de oscilación internos del Sistema de Control Galileo. De esta manera se ahorra tener que estar construyendo oscilaciones. Se dispone de 32 bits de oscilación que varían con la frecuencia del tiempo de ciclo del Sistema de Control Galileo.

Retorno

`true`
Está activo dicho bit
`false`
No está activo dicho bit

Parámetros

1. `dword Bit`
Número de bit de oscilación a consultar. Dicho número será de 0 a 31.

6

```
Machine GetMachineByName(string MachineName)
```

Descripción general

Este método sirve para obtener la máquina que tiene el nombre de máquina indicado en el parámetro *MachineName*.
Como se puede apreciar el método devuelve un componente. Este componente no puede ser memorizado en una variable intermedia, pero puede ser utilizado para invocar algunos de sus métodos.

Retorno

`Machine`
Retorna un componente del Tipo Machine para poder operar con el.
NOTA: En el caso de que el número de máquina no exista se producirá una parada de emergencia del Sistema Galileo y el programa de control se detendrá. Es por lo tanto recomendable comprobar previamente la existencia de dicho número de máquina a través de alguno de los métodos disponibles a tal efecto.

Parámetros

1. `string MachineName`
Nombre de la máquina que queremos obtener.

7

```
Machine GetMachineByNumber(dword MachineNumber)
```

Descripción general

Este método sirve para obtener la máquina que tiene el número de máquina indicado en el parámetro `MachineNumber`.
Como se puede apreciar el método devuelve un componente. Este componente no puede ser memorizado en una variable intermedia, pero puede ser utilizado para invocar algunos de sus métodos.

Retorno

`Machine`
Retorna un componente del `MachineNumber` para poder operar con el.
NOTA: En el caso de que el número de máquina no exista se producirá una parada de emergencia del Sistema Galileo y el programa de control se detendrá. Es por lo tanto recomendable comprobar previamente la existencia de dicho número de máquina a través de alguno de los métodos disponibles a tal efecto.

Parámetros

1. `dword MachineNumber`
   Número de la máquina que queremos obtener.

8

```
bool IsDigit(dword Number)
```

Descripción general

Este método nos indica si un número representando el código ASCII de un carácter corresponde con un dígito (0 a 9) o no. Tiene especial interés para recorrer buffers de lectura (de escaners, custom_data, etc) y poder identificar donde comienza una lectura.

Retorno

`true`
El número de código ASCII corresponde a un dígito
`false`
El número de código ASCII NO corresponde a un dígito

Parámetros

1. `dword Number`
   Valor del código ASCII a consultar

9

```
bool IsErrorActive(dword ErrorNo)
```

Descripción general

Este método sirve para averiguar si un determinado error se encuentra activo.

Retorno

`true`
El error indicado se encuentra activo
`false`
El error indicado no está activo

Parámetros

1. `dword ErrorNo`
   Número de error que se quiere averiguar si está activo.

10

```
bool IsLower(dword Number)
```

Descripción general

Este método nos indica si un número representando el código ASCII de un carácter corresponde con una letra en minúsculas. Tiene especial interés para recorrer buffers de lectura (de escaners, custom_data, etc.)

Retorno

`true`
El número de código ASCII corresponde a una letra en minúsculas.
`false`
El número de código ASCII NO corresponde a una letra en minúsculas.

Parámetros

1. `dword` Number
   Valor del código ASCII a consultar

11

```
bool IsSpace(dword Number)
```

Descripción general

Este método nos indica si un número representando el código ASCII de un carácter corresponde con un espacio. Tiene especial interés para recorrer buffers de lectura (de escaners, custom_data, etc.)

Retorno

`true`
El número de código ASCII corresponde a un espacio.
`false`
El número de código ASCII NO corresponde a un espacio.

Parámetros

1. `dword` Number
   Valor del código ASCII a consultar

12

```
bool IsUpper(dword Number)
```

Descripción general

Este método nos indica si un número representando el código ASCII de un carácter corresponde con una letra en mayúsculas. Tiene especial interés para recorrer buffers de lectura (de escaners, custom_data, etc.)

Retorno

`true`
El número de código ASCII corresponde a una letra en mayúsculas.
`false`
El número de código ASCII NO corresponde a una letra en mayúsculas.

Parámetros

1. `dword` Number
   Valor del código ASCII a consultar

13

```
bool NegativeSignalChange(bool &Signal)
```

Descripción general

Este método es análogo al `SignalChange`, tanto en funcionamiento como en valores retornados, con el matiz de que sólo sirve para detectar flancos descendentes en los cambios de valor de la variable.

Retorno

`true`
Se ha producido un flanco descendente en la variable monitorizada.
`false`
No se ha producido cambio de flanco con respecto a la última vez que se preguntó por el valor de dicha variable.

Parámetros

1. `bool` `Signal`
   Necesita la dirección de la variable de tipo booleano que se que quiere monitorizar para detectar sus flancos negativos.

14

```
dword NumToASCII(dword Number)
```

Descripción general

Este método realiza la conversión de un dígito a su correspondiente código ASCII. Esta especialmente pensado para escritura en buffers de texto.

Retorno

`dword`
Valor del código ASCII que corresponde a dicho dígito.

Parámetros

1. `dword` `Number`
   Valor del número que se desea convertir a ASCII. Debe de tener un valor entre 0 y 9

15

```
bool PositiveSignalChange(bool &Signal)
```

Descripción general

Este método es análogo al `SignalChange`, tanto en funcionamiento como en valores retornados, con el matiz de que sólo sirve para detectar flancos ascendentes en los cambios de valor de la variable.

Retorno

`true`
Se ha producido un flanco ascendente en la variable monitorizada.
`false`
No se ha producido cambio de flanco con respecto a la última vez que se preguntó por el valor de dicha variable.

Parámetros

1. `bool` `Signal`
   Necesita la dirección de la variable de tipo booleano que se que quiere monitorizar para detectar sus flancos positivos.

16

```
void ResetAllFailures()
```

Descripción general

Este método resetea a efectos de la visualización todos los fallos de la instalación.

Retorno

`void`

Este método no retorna nada

Parámetros

Este método no recibe parámetros de entrada

17

```
void ResetFailure(dword Failure,
                  dword Machine)
```

Descripción general

Elimina un fallo en el sistema.
El método solamente elimina la avería a efectos de que la visualización la muestre.

Retorno

`void`

Este método no retorna nada

Parámetros

1. `dword Failure`
   Número de fallo que se desea eliminar
2. `dword Machine`
   Número de máquina que elimina el fallo.
   Si el fallo es global debe de ser 0.

18

```
void ResetGlobalFailures()
```

Descripción general

Este método resetea a efectos de la visualización todos los fallos generales.
Se definen como fallos generales o globales los que no tienen nada que ver con los transportadores.

Retorno

`void`

Este método no retorna nada

Parámetros

Este método no recibe parámetros de entrada

19

```
void ResetMachineFailure(dword Machine)
```

Descripción general

Este método elimina a efectos de la visualización todas las averías de una máquina.
Este método debe asociarse a las acciones de rearme.

Retorno

`void`

Este método no retorna nada

Parámetros

1. `dword Machine`
   Número de máquina que se desea eliminar el defecto.

20

```
void SetCancelEventResponse(dword TransportNumber,
                            word Status)
```

Descripción general

Responde a una cancelación de transporte por parte del SGA

Retorno

`void`

Este método no retorna nada

Parámetros

1. `dword TransportNumber`
   Número de transporte que se desea cancelar.
2. `word Status`
   Respuesta desde el Sistema de Control Galileo a la petición de cancelación:
   a. 0 → No es posible la cancelación (si no se va a cancelar no es necesario invocar este método)
   b. 1 → Cancelado

21

```
void SetFailure(dword Failure,
                dword Machine)
```

Descripción general

Establece un fallo en el sistema.
El método solamente registra una avería a efectos de que la visualización la muestre.

Retorno

`void`

Este método no retorna nada

Parámetros

1. `dword Failure`
   Número de fallo que se desea notificar.
2. `dword Machine`
   Número de máquina que notifica el fallo. Si el fallo es global debe de ser 0

22

```
void SetRouteStatus(byte SourceStationType,
                    byte SourceStationNumber,
                    byte TargetStationType,
                    byte TargetStationNumber,
                    byte Status)
```

Descripción general

Actualiza el estado de una determinada ruta en la base de datos

Retorno

`void`

Este método no retorna nada

Parámetros

1. `byte SourceStationType`
   Tipo de estación origen de la ruta a evaluar.

2. byte SourceStationNumber
Número de la estación origen de la ruta a evaluar.
3. byte TargetStationType
Tipo de estación destino de la ruta a evaluar.
4. byte TargetStationNumber
Número de estación destino de la ruta a evaluar.
5. byte Status
Valor del estado a establecer.

23

```
bool SignalChange(bool &Signal)
```

Descripción general

Este método sirve para detectar un flanco ascendente o descendente en el valor de una variable de tipo global o mapeada en la periferia.
Es importante hacer notar que cuando se detecta un flanco, este es detectado durante todo un ciclo de ejecución del programa. Asimismo, ha re resaltarse la importancia de no intentar detectar cambios de flanco en variables locales a una función.

Retorno

true
Se ha producido un flanco ascendente o descendente en la variable monitorizada.
false
Se ha producido cambio de flanco con respecto a la última vez que se preguntó por el valor de dicha variable.

Parámetros

1. bool Signal
Necesita la dirección de la variable de tipo booleano que se que quiere monitorizar para detectar sus flancos.

24

```
bool TransportHasCancelEvent(dword Transport)
```

Descripción general

Este método permite saber si el SGA ha solicitado la cancelación de un determinado transporte.

Retorno

true
Existe orden de cancelación.
false
NO existe orden de cancelación.

Parámetros

1. dword Transport
Valor del número que se desea convertir a ASCII. Debe de tener un valor entre 0 y 9

25

```
void TransportCancelEventFinished(dword Transport)
```

Descripción general

Con este método se informa al Sistema de Control Galileo que un transporte para el que el SGA había solicitado su cancelación, ha sido ya cancelado.

Retorno

`void`

Este método no retorna nada

Parámetros

1. `dword Transport`
   Número de transporte que se ha tratado de cancelar.

26

```
void TriggerWatchDog()
```

Descripción general

Este método dispara una señalización al perro guardián, con lo cual el tiempo transcurrido de perro guardián se reinicia.
Este método solamente se debe emplear en situaciones absolutamente contadas.

Retorno

`void`

Este método no retorna nada

Parámetros

Este método no recibe parámetros de entrada

27

```
void WriteLog(dword d1,
              dword d2,
              dword d3,
              dword d4)
```

Descripción general

Con este método se escribe una línea de log en el archivo custom.log (c:\Galileo\Logs). La línea contiene la hora y fecha actual, y los datos que se le pasan como parámetros. Cada campo de la línea está separado por el carácter ':'. Este método permite almacenar en un fichero formateado el valor de 4 valores, y posteriormente analizarlo en una hoja de cálculo.

Retorno

`void`

Este método no retorna nada

Parámetros

1. `dword d1 ….d4`
   Valores que se desea almacenar.

4.2 Librería de componentes de acceso al hardware (ComHard.dll)

Para realizar tareas complejas repetitivas el Sistema de Control Galileo dispone del concepto de componente. Dado que el Hardware con el que se trata en las instalaciones es limitado, se ha desarrollado una biblioteca básica de componentes de acceso al hardware que incorpora los elementos más utilizados actualmente. Esta biblioteca puede evolucionar y crecer en el futuro con las mejoras de la tecnología y la implantación de nuevos elementos.

4.2.1 PILZ: PANEL OPERADOR PXT305IBS

4.2.1.1 Interface

El propósito de este apartado es explicar como se realiza la programación de este terminal de operador, que en este caso es PX(T) 305 IBS, así como las prestaciones y el funcionamiento del terminal.

Este panel tiene 4 líneas de cristal líquido (LC-Display) con 20 caracteres, 5 teclas de función con un segundo nivel de control (F1...F10), 2 teclas especiales también con dos niveles (S1...S4), 18 numéricas y de control y 8 leds. La memoria de la PX(T) es una flash-EPROM.

Este panel es un elemento de la periferia distribuida Interbus-S.

En cada mensaje o pantalla se pueden insertar variables con un máximo de 8, concretamente de 0 a 7. Los valores de estas variables pueden ser modificados desde el teclado y enviados a Galileo o pueden ser modificados en el Galileo y enviadas al visualizador, estas son las denominadas *Set value*. Hay otro tipo llamadas *True value* que son modificables únicamente desde Galileo. Además podemos distinguirlas por su tamaño en tres tipos: 8 bits, 16 bits, 32 bits.

El interface del componente es el siguiente:

Class	PXT_305_Interbus
BUS IN	8 bytes
BUS OUT	8 bytes

Listado de los métodos:

1	word AlarmCount()
2	bool Busy()
3	void ClearAlarm(byte AlarmNumber)
4	void ClearAlarms()
5	bool ClearDisplay()
6	void ClearTransportAlarm(byte Number)
7	void ClearTransportAlarms()
8	void ClearWarning(byte WarningNumber)
9	void ClearWarnings()
10	bool DisplayVersionNumber()
11	bool Error()
12	dword GetCurrentMachineNumber()
13	dword GetNextMachineNumber()

14	dword GetPreviousMachineNumber()
15	dword GetScreenForType(dword Type)
16	dword GetScreenNumber()
17	bool IsF1() bool IsF2() bool IsF3() bool IsF4() bool IsF5() bool IsF6() bool IsF7() bool IsF8() bool IsF9() bool IsF10() bool IsS1() bool IsS2() bool IsS3() bool IsS4()
18	bool IsMachineHandled(dword Machine)
19	bool LedOff(dword LedNumber)
20	bool LedOn(dword LedNumber)
21	void Load(string File)
22	dword RecvKeyCode()
23	bool SendBinMessage(word nNumber, bool hCopy)
24	bool SendVariable16(word varNumber, byte Value)
25	bool SendVariable32(word varNumber, byte Value)
26	bool SendVariable8(word varNumber, byte Value)
27	void SetAlarm(word AlarmNumber)
28	bool SetCursorPosition(byte X, byte Y)
29	void SetTransportAlarm(dword Offset, dword Message, dword ConveyorNumber)
30	void SetVarForAllMachines(string Variable, dword Value)
31	void SetWarning(word WarningNumber)
32	word TransportAlarmCount()
33	word WarningCount()

Descripción de los métodos:

1

```
word AlarmCount()
```

Descripción general

Este método sirve para saber la cantidad de alarmas insertadas en la cola de

alarmas.

Retorno

word

Retorna la cantidad de alarmas insertadas en la cola.

Parámetros

Este método no tiene parámetros de entrada.

2

```
bool Busy()
```

Descripción general

Retorna un booleano indicando si hay comandos en la pila o no, esto indica que el objeto tiene aún comandos por despachar.

Retorno

true

Hay comandos pendientes de despachar en la pila.

false

NO hay comandos pendientes de despachar en la pila.

Parámetros

Este método no tiene parámetros de entrada.

3

```
void ClearAlarm(byte AlarmNumber)
```

Descripción general

Este método saca de la cola de alarmas un número determinado de alarma, los números de alarma van entre 0 y 9.

Retorno

void

Este método no retorna ningún valor.

Parámetros

1. byte AlarmNumber
   Número de mensaje de alarma a extraer de la pila.

4

```
void ClearAlarms()
```

Descripción general

Este método saca todas las alarmas de la cola de alarmas, con lo cual se dejan de visualizar las mismas.

Retorno

void

Este método no retorna ningún valor.

Parámetros

Este método no tiene parámetros de entrada.

5

```
bool ClearDisplay()
```

Descripción general

Esta función limpia el contenido del display.

Retorno

`true`
Se ha conseguido introducir el comando en la pila.
`false`
NO se ha conseguido introducir el comando en la pila

Parámetros

Este método no tiene parámetros de entrada.

6

```
void ClearTransportAlarm(byte Number)
```

Descripción general

Este método elimina de la cola de alarmas de transportador una alarma determinada.

Retorno

`void`
Este método no retorna nada.

Parámetros

1. `byte` `Number`
   Número de alarma a eliminar de la pila.

7

```
void ClearTransportAlarms()
```

Descripción general

Este método elimina todas las alarmas de transportadores de la pila de alarmas.

Retorno

`void`
Este método no retorna nada.

Parámetros

Este método no tiene parámetros de entrada.

8

```
void ClearWarning(byte WarningNumber)
```

Descripción general

Este método saca de la cola de avisos un número determinado de aviso los números de aviso van entre 0 y 9.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `word` `WarningNumber`
Número de mensaje de Warning a extraer de la pila.

9

```
void ClearWarnings()
```

Descripción general

Este método saca todos los avisos de la cola de avisos, con lo cual se dejan de visualizar los mismos.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

Este método no tiene parámetros de entrada.

10

```
bool DisplayVersionNumber()
```

Descripción general

Esta función hace que se muestre en el display su versión de firmware actual.

Retorno

`true`
Se ha conseguido introducir el comando en la pila.
`false`
NO se ha conseguido introducir el comando en la pila

Parámetros

Este método no tiene parámetros de entrada.

11

```
bool Error()
```

Descripción general

Retorna un booleano indicando si el visualizador se encuentra en error o no.

Retorno

`true`
El elemento se encuentra en error.
`false`
El elemento no se encuentra en condición de error.

Parámetros

Este método no tiene parámetros de entrada.

12

```
dword GetCurrentMachineNumber()
```

Descripción general

El visualizador permite seleccionar máquinas (transportadores) e ir pasando de uno a otro a través de la pulsación siguiente o anterior. Este método nos permite saber cual es el actual transportador seleccionado.

Retorno

dword

Retorna el número del transportador actual seleccionado.

Parámetros

Este método no tiene parámetros de entrada

13

```
dword GetNextMachineNumber()
```

Descripción general

El visualizador permite seleccionar máquinas (transportadores) e ir pasando de uno a otro a través de la pulsación siguiente o anterior. Este método nos permite saber cual es el posterior transportador configurado.

Retorno

dword

Retorna el número del transportador posterior al actual según el archivo de configuración.

Parámetros

Este método no tiene parámetros de entrada

14

```
dword GetPreviousMachineNumber()
```

Descripción general

El visualizador permite seleccionar máquinas (transportadores) e ir pasando de uno a otro a través de la pulsación siguiente o anterior. Este método nos permite saber cual es el anterior transportador configurado.

Retorno

dword

Retorna el número del transportador anterior al actual según el archivo de configuración.

Parámetros

Este método no tiene parámetros de entrada

15

```
dword GetScreenForType(dword Type)
```

Descripción general

Este método sirve para saber la pantalla que se ha configurado en el archivo de configuración del visualizador como pantalla para controlar dicho Tipo de máquina.

Retorno

dword

Retorna el número de pantalla seleccionada para realizar el control.

Parámetros

1. `string` Variable
Número que contiene el tipo de máquina a controlar.

16

```
dword GetScreenNumber()
```

Descripción general

Este método sirve para obtener el número de pantalla (también llamado número de mensaje) que se encuentra activa actualmente en el visualizador.

Retorno

`dword`
Retorna el número de pantalla que se está mostrando en el visualizador en ese momento.

Parámetros

Este método no tiene parámetros de entrada.

17

```
bool IsF1()
bool IsF2()
bool IsF3()
bool IsF4()
bool IsF5()
bool IsF6()
bool IsF7()
bool IsF8()
bool IsF9()
bool IsF10()
bool IsS1()
bool IsS2()
bool IsS3()
bool IsS4()
```

Descripción general

El visualizador tiene una característica especial que afecta solamente a las teclas de función. Estas teclas tienen una imagen de pulsación directamente sobre periferia. El Componente expone un conjunto de métodos destinados a saber si por parte de la periferia hay constancia de pulsación. Este método es el que se debe emplear para manejar controles en manual (mover un transportador en manual por ejemplo)
NOTA: Los métodos de recibir tecla funcionan por mensajería y por lo tanto no son seguros para hacer depender el paro o marcha de un transportador de ellos cualquier accionamiento manual debería depender de estos métodos. Por el contrario no sirven para evaluar flancos, ya que no indican cambios de estado si no estado de la pulsación de una tecla por nivel.

Retorno

`true`
Si la tecla indicada se encuentra pulsada.
`false`
Si la tecla indicada NO está pulsada.

Parámetros

Este método no tiene parámetros de entrada.

18

```
bool IsMachineHandled(dword Machine)
```

Descripción general

Este método indica si un número de máquina está incluido en el fichero de configuración de este panel, es decir, si esa máquina puede ser tratada desde ese terminal.

Retorno

`true`
Si dicha máquina está especificada en la configuración del panel.
`false`
Si dicha máquina NO está especificada en la configuración del panel.

Parámetros

1. `dword` Machine
   Número de máquina de la que se quiere saber si es o no manejada desde el panel.

19

```
bool LedOff(dword LedNumber)
```

Descripción general

Este método permite desactivar un LED determinado de los que se encuentran en las teclas del visualizador.

Retorno

`true`
Si el número de LED indicado es correcto.
`false`
Si el número de LED indicado NO es correcto.

Parámetros

1. `dword` LedNumber
   Indica el número de LED que se desea activar. En el visualizador que nos ocupa este parámetro puede tomar valores 0 a 4 para activar los LEDs de las teclas de función F1 a F5 o bien tomar valores 12 y 13 para activar los LEDs correspondientes a las teclas de función S1 y S2.

20

```
bool LedOn(dword LedNumber)
```

Descripción general

Este método permite activar un LED determinado de los que se encuentran en las teclas del visualizador.

Retorno

`true`
Si el número de LED indicado es correcto.
`false`

Si el número de LED indicado NO es correcto.

Parámetros

1. `dword` `LedNumber`
   Indica el número de LED que se desea activar. En el visualizador que nos ocupa este parámetro puede tomar valores 0 a 4 para activar los LEDs de las teclas de función F1 a F5 o bien tomar valores 12 y 13 para activar los LEDs correspondientes a las teclas de función S1 y S2.

21

```
void Load(string File)
```

Descripción general

Este método carga el archivo un archivo de configuración. Solamente se puede mantener cargado e memoria un archivo de configuración con lo cual si se invoca dos veces este método se descargará el primer archivo cargado. Este método está diseñado para ejecutarse desde las funciones de arranque frío o caliente. En ningún caso debería ejecutarse en el código cíclico.
El formato de este archivo se explicará después.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `string` `File`
   Nombre del archivo a cargar.

22

```
dword RecvKeyCode()
```

Descripción general

Este método permite obtener la última tecla pulsada en el visualizador.

Retorno

`dword`
Retorna en el valor numérico de la tecla pulsada (para saber cuales son los valores numéricos de cada una de las teclas debe consultarse la documentación del visualizador). Tiene además la propiedad de que cada vez que se invoca el método y había una tecla pulsada se elimina la misma de los valores memorizados, con lo cual el método también sirve como detector de flanco de pulsación.

Parámetros

Este método no tiene parámetros de entrada.

23

```
bool SendBinMessage(word nNumber,
                    bool hCopy)
```

Descripción general

Este método indica al visualizador que muestre un mensaje determinado pudiendo

indicar a su vez si se desea copia impresa del mismo o no.

Retorno

`true`
Se ha conseguido introducir el comando en la pila.
`false`
NO se ha conseguido introducir el comando en la pila

Parámetros

1. `word` nNumber
   Número del mensaje que se desea visualizar.
2. `bool` hCopy
   Será `true` si se desea copia impresa o `false` si no se desea realizar copia impresa.

24

```
bool SendVariable16(word varNumber,
                    byte Value)
```

Descripción general

Esta función establece el valor de una variable configurada en el PXT como variable de tipo byte (16 bits).

Retorno

`true`
Se ha conseguido introducir el comando en la pila.
`false`
NO se ha conseguido introducir el comando en la pila

Parámetros

1. `word` varNumber
   Número de la variable a establecer (0 a 7).
2. `byte` Value
   Valor que se desea asignar a la variable.

25

```
bool SendVariable32(word varNumber,
                    byte Value)
```

Descripción general

Esta función establece el valor de una variable configurada en el PXT como variable de tipo byte (32 bits).

Retorno

`true`
Se ha conseguido introducir el comando en la pila.
`false`
NO se ha conseguido introducir el comando en la pila

Parámetros

1. `word` varNumber
   Número de la variable a establecer (0 a 7).
2. `byte` Value
   Valor que se desea asignar a la variable.

26

```
bool SendVariable8(word varNumber,
                   byte Value)
```

Descripción general

Esta función establece el valor de una variable configurada en el PXT como variable de tipo byte (8 bits).

Retorno

`true`
Se ha conseguido introducir el comando en la pila.
`false`
NO se ha conseguido introducir el comando en la pila

Parámetros

1. `word` `varNumber`
   Número de la variable a establecer (0 a 7).
2. `byte` `Value`
   Valor que se desea asignar a la variable.

27

```
void SetAlarm(word AlarmNumber)
```

Descripción general

Este método establece una alarma en el sistema. Una alarma es un evento de prioridad máxima, cuando un evento de este tipo se produce el visualizador pasa a presentar la información del mismo en la pantalla, es decir el visualizador cambia el mensaje al número de alarma presentado en el parámetro indicado. Cuando este método es invocado el visualizador encola la alarma en una cola de alarmas. Mientras haya algo en la cola de alarmas no se mostrarán más mensajes que los de alarma. Estos mensajes se mostrarán de forma iterativa en el visualizador hasta que desaparezcan. Solamente desaparecen por código mediante la invocación de los métodos correspondientes.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `word` `AlarmNumber`
   Número de mensaje de alarma a visualizar.

28

```
bool SetCursorPosition(byte X,
                       byte Y)
```

Descripción general

Establece la posición del cursor (el punto a partir del cual se escribirá en el visualizador) en las coordenadas X e Y deseadas.

Retorno

`true`
Se ha conseguido introducir el comando en la pila.
`false`
NO se ha conseguido introducir el comando en la pila

Parámetros

1. `byte X`
Coordenada X deseada.
2. `byte Y`
Coordenada Y deseada.

29

```
void SetTransportAlarm(dword Offset,
                       dword Message,
                       dword ConveyorNumber)
```

Descripción general

Este método establece una alarma de transportador. Estas alarmas funcionan como las alarmas normales o convencionales pero están especialmente preparadas para su uso con transportadores.

Retorno

`void`
Este método no retorna nada.

Parámetros

1. `dword Offset`
Mensaje a partir del cual se sitúan los mensajes.
2. `dword Message`
Número de mensaje a partir del offset establecido.
3. `dword ConveyorNumber`
Número de transportador que genera la avería.

30

```
void SetVarForAllMachines(string Variable,
                          dword Value)
```

Descripción general

Este método sirve para comandar simultáneamente el cambio de una variable para todas las máquinas configuradas como controladas desde este visualizador. Por ejemplo su utilización es para eliminar el manual simultáneamente en todos los transportadores de la zona controlada por dicho visualizador.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `string Variable`
Nombre de la variable que se desea manipular en todas las máquinas
2. `dword Value`
Valor que se desea asignar a la variable en todas las máquinas.

31

```
void SetWarning(word WarningNumber)
```

Descripción general

Este método establece un aviso en el sistema. Un aviso es un evento de prioridad media, cuando un evento de este tipo se produce el visualizador pasa a presentar la información del mismo en la pantalla es decir el visualizador cambia el mensaje al número de aviso presentado en el parámetro indicado si no hay alarmas pendientes. Cuando este método es invocado el visualizador encola los avisos en una cola. Mientras haya algo en la cola de alarmas no se mostrarán más mensajes que los de alarma. Cuando la cola de alarmas esté vacía se mostrarán los avisos. Estos mensajes se mostrarán de forma iterativa en el visualizador hasta que desaparezcan. Solamente desaparecen por código mediante la invocación de los métodos correspondientes.

Retorno

```
void
```

Este método no retorna ningún valor.

Parámetros

1. `word` `WarningNumber`
   Número de mensaje de Warning a visualizar.

32

```
word TransportAlarmCount()
```

Descripción general

Este método sirve para saber la cantidad de alarmas de transportador insertadas en la cola de alarmas.

Retorno

```
word
```

Retorna la cantidad de alarmas insertadas en la cola.

Parámetros

Este método no tiene parámetros de entrada.

33

```
word WarningCount()
```

Descripción general

Este método sirve para saber la cantidad de avisos insertados en la cola de avisos.

Retorno

```
word
```

Retorna la cantidad de avisos insertados en la cola.

Parámetros

Este método no tiene parámetros de entrada.

4.2.1.2 El fichero de configuración de mensajes

El sistema implementado es capaz de gestionar cierta lógica de transferencia de variables desde las pantallas de visualización hacia el Sistema de Control Galileo y viceversa sin requerir ni una sola línea de código. Esto se logra cargando en el arranque del programa Galileo en el componente de la Pilz un archivo de

configuración donde se le indica la cantidad de variables que tienen los diferentes mensajes, el tipo y a qué variable interna de Galileo se desea enlazar.

Pese a que el visualizador permite la creación de hasta 10000 mensajes se ha restringido la cantidad de entradas gestionadas por Galileo de forma automática a 128, este número es una convención y puede ser incrementado en el futuro si resulta inadecuado. Hay que hacer notar que sólo los mensajes que tienen variables que se desean vincular con el bus deben ser introducidos en el archivo de configuración, con lo cual la limitación de 128 no es tal limitación ya que no todos los mensajes (ni mucho menos) tienen variables vinculadas.

A continuación se muestra la estructura del archivo indicando en gris comentarios explicativos.

```
; REGLAS BASICAS
; Los archivos INI admiten como comentario cualquier linea que
comience con ;el simbolo de punto y coma
; no es conveniente poner un comentario en una linea con datos
;La sección de cabecera cotiene los datos generales a todo el INI

[header]
; El campo mensajes contiene las pantallas asociadas al
visualizador
;PXT_305 que llevan algún tipo de autogestión manual

  mensajes=3,12,13,14,15,16,17,18,19

  ; El campo maquinas contiene la lista de máquinas que se asocia
a este
  ; visualizador. El visualizador permitirá avanzar entre ellas
hacia
  ; delante o hacia atrás

  maquinas=1,2,3,4,5,6,7,8,9,10

; Por cada pantalla que aparezca en la entrada de mensajes debe ;
; aparecer
; aqui una sección con dicho número
; estas definiciones de pantallas permiten realizar las siguientes
; funciones:
; 1°- Asociar Variables a variables internas de la pantalla PILZ a
; través de
;     Var(x) y Tipo(x)
; 2°- Asociar funcionamientos seguimientos de variables
automáticos es decir
;     al pulsar cualquier tecla de función ( F1 a F10 ) o
;     las teclas auxiliares S1 a S4 se puede asociar una variable
; para que
;     se fuerce y por lo tanto realice funcionamiento automático.

[3]
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
[12]
```

```
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
  F1      = M_AdelanteManual
  F3      = M_SiguienteManual
  F5      = M_AtrasManual
  VarMaquina = PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
[13]
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
  F1      = M_AdelanteManual
  F3      = M_SiguienteManual
  F5      = M_AtrasManual
  VarMaquina = PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
[14]
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
  F1     = M_AdelanteManual
  F3     = M_SiguienteManual
  F5     = M_AtrasManual
  S1     = M_SubirManual
  S2     = M_BajarManual
  VarMaquina = PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
[15]
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
  F1     = M_AdelanteManual
  F2     = M_AdelanteManual
  F3     = M_SiguienteManual
  F4     = M_AtrasManual
  F5     = M_AtrasManual
  S1     = M_SubirManual
  S2     = M_BajarManual
  VarMaquina = PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
[16]
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
  F1     = M_AdelanteManual
  F2     = M_AntihorarioManual
  F3     = M_SiguienteManual
  F4     = M_HorarioManual
  F5     = M_AtrasManual
  S1     = M_SubirManual
  S2     = M_BajarManual
  VarMaquina = PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
[17]
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
  F1     = M_AdelanteManual
  F3     = M_SiguienteManual
  F5     = M_AtrasManual
  S1     = M_AbrirManual
  S2     = M_CerrarManual
  VarMaquina = PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
[18]
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
```

```
  F1      = M_AdelanteManual
  F2      = M_IzquierdaManual
  F3      = M_SiguienteManual
  F4      = M_DerechaManual
  F5      = M_AtrasManual
  VarMaquina = PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
[19]
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
  F1      = M_AdelanteManual
  F2      = M_IzquierdaManual
  F3      = M_SiguienteManual
  F4      = M_DerechaManual
  F5      = M_AtrasManual
  S1      = M_AbrirManual
  S2      = M_CerrarManual
  VarMaquina = PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz

; Esta sección define asociaciones entre el valor del TipoManual
de cada
; máquina y cual es la pantalla
; que realizará el manejo de la mima.
; Cada tipo de movimiento se configura mediante un bit:
; Bit0: Adelante Rodillos
; Bit1: Atras    Rodillos
; Bit2: Adelante Cadenas
; Bit3: Atras    Cadenas
; Bit4: Subir
; Bit5: Bajar
; Bit6: Giro     Horario
; Bit7: Giro     Antihorario
; Bit8 a 31: Definidos por usuario
; El Primer número en el TipoManual consiste en el valor decimal
; correspondiente
; Ej: 3 = 12 quiere decir que como 3 en binario son los bit 0 y 1
activos el
; transportador soporta adelante / atras rodillos
; y que en el caso en el que la máquina a controlar sea de este
tipo se
; enviará al usuario a la pantalla 50

[TiposManual]
  1 = 50
  3 = 50
  51 = 52
  63 = 53
  255 = 54
  771 = 55
  3075 = 56
  3843 = 57
```

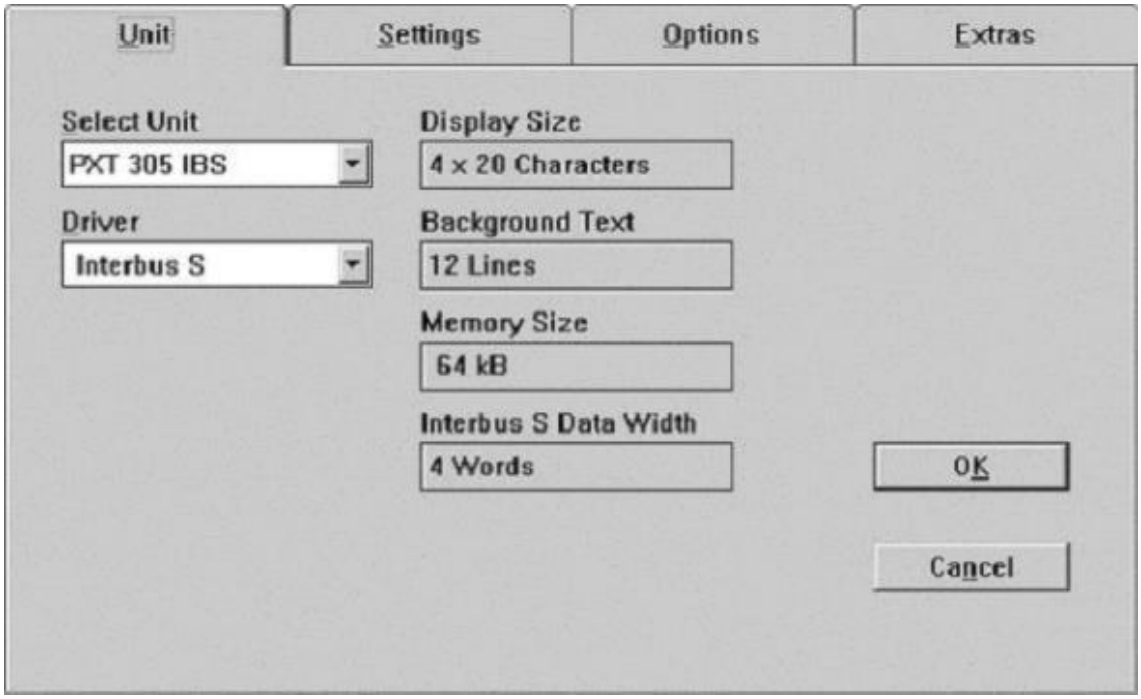
En todo caso los nombres de variables que se indican deben corresponder al nombre de una de las variables que existen en el programa Xana.

Los nombres de las variables están limitados a 200 caracteres.

4.2.1.3 Configuración del Visualizador

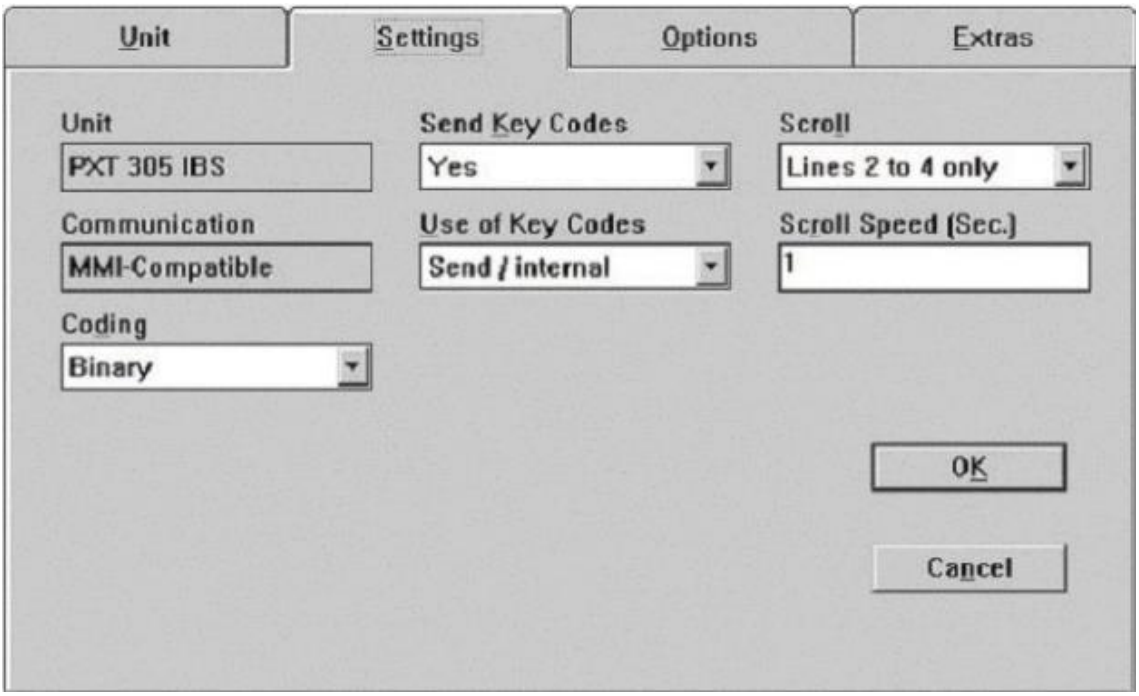
El primer paso en la parametrización de mensajes de la PX(T) es, una vez ejecutado el programa, abrir un nuevo fichero (*File → New*). Se muestran 4 fichas de configuración:

- En la pestaña *Unit* hay que seleccionar el tipo de terminal que se está usando, en nuestro caso PXT 305 IBS, el driver será Interbus-S ya que como se ha dicho antes este terminal está diseñado para trabajar en un bus Interbus.



El resto de los parámetros de esta pestaña no pueden ser modificados.

- En la pestaña ***settings*** los parámetros son:

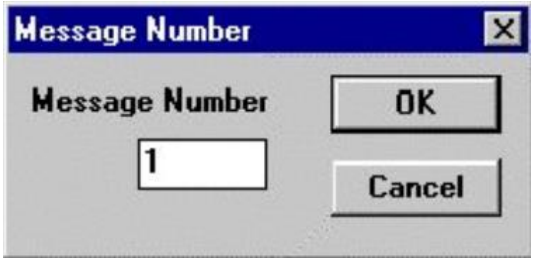


- Coding: Envía el número de la pantalla en binario al sistema de control.
- Send Key Codes**:** Envía el número de tecla pulsada al PLC. Es necesario seleccionar esta opción para que la PX(T) haga un uso interno de las pulsaciones de las teclas, por ejemplo para el scroll.
- Scroll**:** Dependiendo de la aplicación del terminal se selecciona que el scroll sea de las 4 líneas del display, de la línea 2 a la 4, de la 3 y la 4 o sólo de la 4. (En el caso del pupitre de una cabecera con un autómata S7, el scroll será de las líneas 2, 3 y 4).
- Scroll speed**:** la velocidad del scroll por defecto es 2 pero se puede se ajustada de 1 a 9 segundos.

Los parámetros de las otras dos pestañas se dejan por defecto. Si se quieren visualizar los parámetros o modificar posteriormente alguno de ellos en "windows→display" o pulsar el botón.

4.2.1.3.1 Configuración de los mensajes

Definidos los parámetros del terminal, se comienza a introducir los mensajes o pantallas que van a aparecer en el display del terminal.



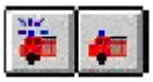
Esta pantalla de selección del número de mensaje aparece la primera vez al aceptar la pantalla de configuración del terminal, con "*insert→message*" o con el botón:

En cada mensaje se puede incluir texto, o definir variables, dependiendo de las necesidades concretas de la aplicación.
En la parte derecha de la pantalla, hay un área para escribir comentarios, estos comentarios no se ven en el terminal en el funcionamiento online. No se puede editar al mismo tiempo el área de variables y el área de comentarios, para habilitar el área de comentarios en "*View→comment*" o pulsando el botón:

El área de comentarios se deshabita "*View→comment*".
Al crear una nueva pantalla, ésta aparece con una única línea, se pueden incluir hasta 16. Para insertar una nueva línea "*Insert→new line*" o con el botón:

Para borrar una línea "*edit→delete line*"" o con el botón:

Se puede hacer que un texto dentro de la pantalla parpadee, para ello seleccionar el texto y con "*insert→ flash ON*". Para quitar el parpadeo "*insert→flash OFF*". O con los botones:



Disponemos de un menú especial (pulsando con el botón derecho del ratón sobre cualquier parte de la hoja de los menús) para moverse de forma rápida de un mensaje a otro o para crear nuevos mensajes. En este menú especial hay otra opción para incluir en el texto de los mensajes caracteres especiales.

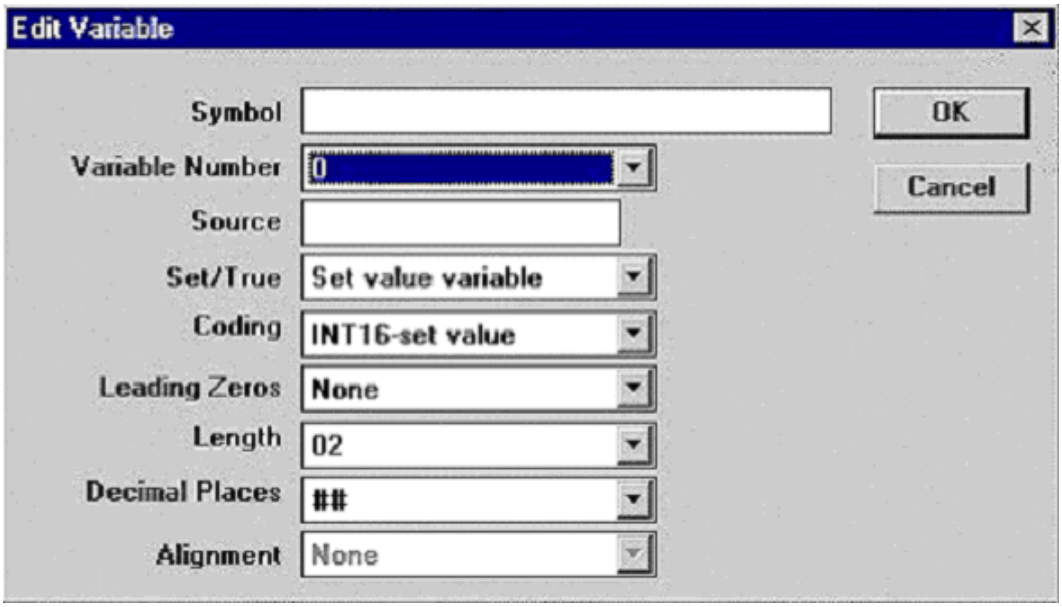
4.2.1.3.2 Variables

Para insertar variable hay que situar el cursor en el punto en el cual se quiere insertar la variable y con "*insert→variable*" o con el botón:

En una pantalla se pueden insertar 8 variables numeradas de 0 a 7. Pueden tener una longitud máxima de 10 caracteres. Las variables pueden ser:

- *True value variable*: Para leer valores del sistema de control (PLC o PC) y ser mostrados en la pantalla.
- *Set value variable*: para leer o escribir (introducidos a través del teclado) valores sobre el sistema de control (PLC o PC) .

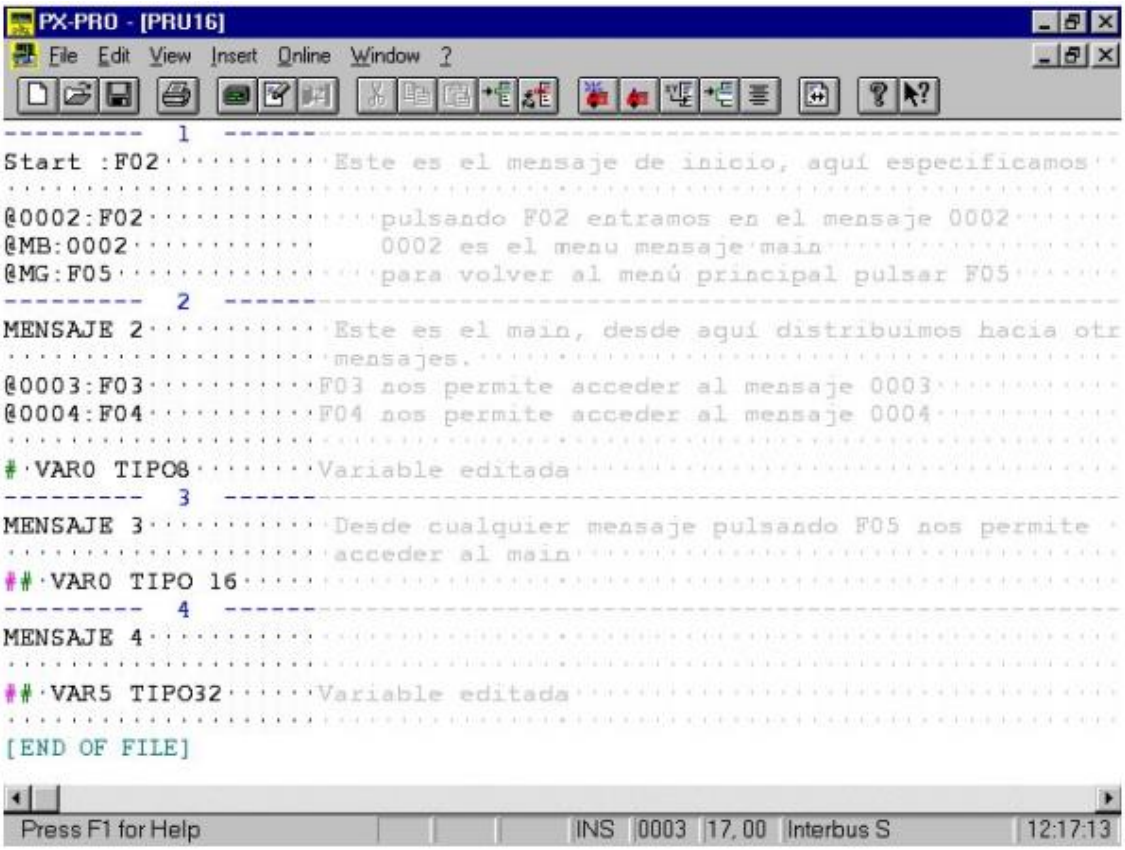
En *coding* seleccionamos el tamaño y el formato de la variable.



Cada dígito de la variable se representa en el PX-PRO con el carácter #. Estos caracteres sólo son visibles en el programa, no en el visualizador.

4.2.1.3.3 Creación de menús de mensajes

Pilz permite crear menús con distintos mensajes y pasar de unos a otros mediante el uso de teclas de funciones. E incluso permite la posibilidad de exigir la petición de passwords para acceder a algunos de estos mensajes. Para definir las teclas de función que producen el paso de unos mensajes a otros se utilizan etiquetas.
Comentemos sobre un ejemplo:



Éste es un sencillo ejemplo de cómo se configuran una estructura de menús permitiendo moverse a través de ella mediante teclas de función.
Podemos definir un menú de passwords de hasta 8 posibles niveles. Para los distintos niveles tendremos acceso a un grupo de mensajes de igual prioridad o inferior. Es decir introduciendo la password de nivel 1, tenemos acceso a todos los menús del visualizador.

Cada password está compuesta por 4 dígitos. El tipo de entrada para definir un menú de password es:

- @Px:#### siendo x=1..8 especificamos el nivel, #### se refiere a los 4 dígitos.

Cuando queremos especificar la necesidad de necesitar una password para acceder a determinado mensaje lo hacemos de la siguiente manera:

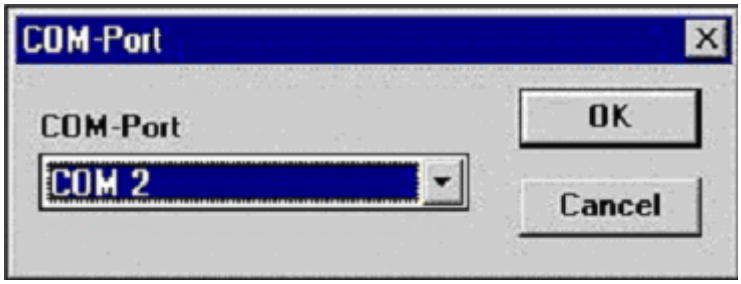
- @yyyy:Fxx,Pz siendo yyyy el número del mensaje a acceder, xx el número de la tecla de función, z el nivel de protección que tenemos sobre el mensaje.

NOTA: Cuando especificamos Fxx nos referimos que la tecla de función lleva dos cifras, es decir cuando nos referimos a la tecla de función nº 5 tenemos que especificarla como F05.

Cuando seleccionamos un menú con password-protected te hará la petición de la misma. En caso de equivocación nos sale un mensaje de error.

4.2.1.3.4 Comunicación

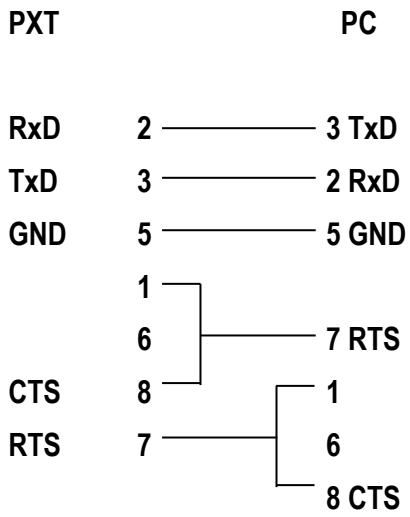
Para la transmisión de los datos a la PX(T) se necesita un cables serie (RS232). Para elegir el puerto de comunicación del PC "Online→select port com":



Para poder transferirle el programa a la PX(T) ésta tiene que estar offline. Esto se consigue de dos formas:

1. Quitar la alimentación y volver a dársela con la tecla F1 pulsada.
2. Pulsar la secuencia de teclas: ENT después flecha hacia arriba después F1.

Desde el programa PX PRO con "*online→write text memory*", copiamos en la memoria flash-EPROM del terminal la configuración, la pantallas y variables.
Cuando finalice la transmisión pulsar cualquier tecla excepto F1.
Cable de conexión PX(T)-PC RS-232 de 9 pines.



4.2.2 EXOR: PANEL OPERADOR UNIOP

4.2.2.1 Interface

El propósito de este apartado es explicar como se realiza la programación de este terminal de operador, UniOP de Exor (o Hitachi), así como las prestaciones y el funcionamiento del terminal.

De este panel existen dos versiones. Una tiene 4 líneas de cristal líquido (LC-Display) con 20 caracteres, 12 teclas de función (F1…F12), 23 numéricas y de control y 12 LEDs. La otra versión dispone de una pantalla táctil STN con una resolución de 320x240, y 32 Mb de memoria. La memoria de estos paneles es flash-EPROM.

Este panel es un elemento de la periferia distribuida Profibus-PB.

Permite la opción de funcionar con 4 ó 16 `words`, pero el ***componente está preparado para funcionar con el perfil de 8 `words` sólamente***.

En cada mensaje o pantalla se pueden insertar variables referidas a una periferia simulada que siempre debe situarse en el ***DB2*** del PLC. El tamaño máximo disponible de esta memoria simulada es de 8192 `bytes`. En el fichero .ini de configuración se explica como ligar estas direcciones virtuales con variables de Galileo, de forma que se actualicen en ambos sentidos sus valores al ser mostrados en el panel del operador. Los valores de estas variables pueden ser modificados desde el teclado y enviados a Galileo o pueden ser modificados en el Galileo y enviadas al visualizador, sin necesidad de ninguna programación extra por parte de control.

El interface utilizado para este dispositivo es el siguiente:

Class	UniOpe
BUS IN	16 bytes
BUS OUT	16 bytes

Listado de métodos:

1	word AlarmCount()
2	bool Busy()
3	void ClearAlarm(byte AlarmNumber)
4	void ClearAlarms()
5	void ClearTransportAlarm(byte Number)
6	void ClearTransportAlarms()
7	void ClearWarning(byte WarningNumber)
8	void ClearWarnings()
9	bool Error()
10	dword GetCurrentMachineNumber()
11	dword GetNextMachineNumber()
12	dword GetPreviousMachineNumber()
13	dword GetScreenForType(dword Type)
14	dword GetScreenNumber()
15	bool IsF1() bool IsF2() bool IsF3()

	bool IsF4() bool IsF5() bool IsF6() bool IsF7() bool IsF8() bool IsF9() bool IsF10() bool IsF11() bool IsF12()
16	bool IsMachineHandled(dword Machine)
17	bool LedOff(dword LedNumber)
18	bool LedOn(dword LedNumber)
19	void Load(string File)
20	dword RecvKeyCode()
21	bool SendBinMessage(word nNumber, bool hCopy)
22	void SetAlarm(word AlarmNumber)
23	bool SetScreenNumber(byte Page)
24	void SetTransportAlarm(dword Offset, dword Message, dword ConveyorNumber)
25	void SetVarForAllMachines(string Variable, dword Value)
26	void SetWarning(word WarningNumber)
27	word TransportAlarmCount()
28	word WarningCount()word

Descripción de los métodos:

1

```
word AlarmCount()
```

Descripción general

Este método sirve para saber la cantidad de alarmas insertadas en la cola de alarmas.

Retorno

word

Retorna la cantidad de alarmas insertadas en la cola.

Parámetros

Este método no tiene parámetros de entrada.

2

```
bool Busy()
```

Descripción general

Retorna un booleano indicando si hay comandos en la pila o no, esto indica que el objeto tiene aún comandos por despachar.

Retorno

`true`
Hay comandos pendientes de despachar en la pila.
`false`
NO hay comandos pendientes de despachar en la pila.

Parámetros

Este método no tiene parámetros de entrada.

3

```
void ClearAlarm(byte AlarmNumber)
```

Descripción general

Este método saca de la cola de alarmas un número determinado de alarma, los números de alarma van entre 0 y 9.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `byte` `AlarmNumber`
   Número de mensaje de alarma a extraer de la pila.

4

```
void ClearAlarms()
```

Descripción general

Este método saca todas las alarmas de la cola de alarmas, con lo cual se dejan de visualizar las mismas.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

Este método no tiene parámetros de entrada.

5

```
void ClearTransportAlarm(byte Number)
```

Descripción general

Este método elimina de la cola de alarmas de transportador una alarma determinada.

Retorno

`void`
Este método no retorna nada.

Parámetros

2. `byte` `Number`
   Número de alarma a eliminar de la pila.

6

```
void ClearTransportAlarms()
```

Descripción general

Este método elimina todas las alarmas de transportadores de la pila de alarmas.

Retorno

`void`

Este método no retorna nada.

Parámetros

Este método no tiene parámetros de entrada.

7

```
void ClearWarning(byte WarningNumber)
```

Descripción general

Este método saca de la cola de avisos un número determinado de aviso los números de aviso van entre 0 y 9.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word` `WarningNumber`
   Número de mensaje de Warning a extraer de la pila.

8

```
void ClearWarnings()
```

Descripción general

Este método saca todos los avisos de la cola de avisos, con lo cual se dejan de visualizar los mismos.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no tiene parámetros de entrada.

9

```
bool Error()
```

Descripción general

Retorna un booleano indicando si el visualizador se encuentra en error o no.

Retorno

`true`

El elemento se encuentra en error.

`false`

El elemento no se encuentra en condición de error.

Parámetros

Este método no tiene parámetros de entrada.

10

```
dword GetCurrentMachineNumber()
```

Descripción general

El visualizador permite seleccionar máquinas (transportadores) e ir pasando de uno a otro a través de la pulsación siguiente o anterior. Este método nos permite saber cual es el actual transportador seleccionado.

Retorno

dword

Retorna el número del transportador actual seleccionado.

Parámetros

Este método no tiene parámetros de entrada

11

```
dword GetNextMachineNumber()
```

Descripción general

El visualizador permite seleccionar máquinas (transportadores) e ir pasando de uno a otro a través de la pulsación siguiente o anterior. Este método nos permite saber cual es el posterior transportador configurado.

Retorno

dword

Retorna el número del transportador posterior al actual según el archivo de configuración.

Parámetros

Este método no tiene parámetros de entrada

12

```
dword GetPreviousMachineNumber()
```

Descripción general

El visualizador permite seleccionar máquinas (transportadores) e ir pasando de uno a otro a través de la pulsación siguiente o anterior. Este método nos permite saber cual es el anterior transportador configurado.

Retorno

dword

Retorna el número del transportador anterior al actual según el archivo de configuración.

Parámetros

Este método no tiene parámetros de entrada

13

```
dword GetScreenForType(dword Type)
```

Descripción general

Este método sirve para saber la pantalla que se ha configurado en el archivo de configuración del visualizador como pantalla para controlar dicho Tipo de máquina.

Retorno

dword

Retorna el número de pantalla seleccionada para realizar el control.

Parámetros

2. string Variable

   Número que contiene el tipo de máquina a controlar.

14

```
dword GetScreenNumber()
```

Descripción general

Este método sirve para obtener el número de pantalla (también llamado número de mensaje) que se encuentra activa actualmente en el visualizador.

Retorno

dword

Retorna el número de pantalla que se está mostrando en el visualizador en ese momento.

Parámetros

Este método no tiene parámetros de entrada.

15

```
bool IsF1()
bool IsF2()
bool IsF3()
bool IsF4()
bool IsF5()
bool IsF6()
bool IsF7()
bool IsF8()
bool IsF9()
bool IsF10()
bool IsF11()
bool IsF12()
```

Descripción general

El visualizador tiene una característica especial que afecta solamente a las teclas de función. Estas teclas tienen una imagen de pulsación directamente sobre periferia. El Componente expone un conjunto de métodos destinados a saber si por parte de la periferia hay constancia de pulsación. Este método es el que se debe emplear para manejar controles en manual (mover un transportador en manual por ejemplo) OJO: Los métodos de recibir tecla funcionan por mensajería y por lo tanto no son seguros para hacer depender el paro o marcha de un transportador de ellos cualquier accionamiento manual debería depender de estos métodos. Por el contrario no sirven para evaluar flancos, ya que no indican cambios de estado si no estado de la pulsación de una tecla por nivel.

Retorno

true

Si la tecla indicada se encuentra pulsada.
`false`
Si la tecla indicada NO está pulsada.

Parámetros

Este método no tiene parámetros de entrada.

16

```
bool IsMachineHandled(dword Machine)
```

Descripción general

Este método indica si un número de máquina está incluido en el fichero de configuración de este panel, es decir, si esa máquina puede ser tratada desde ese terminal.

Retorno

`true`
Si dicha máquina está especificada en la configuración del panel.
`false`
Si dicha máquina NO está especificada en la configuración del panel.

Parámetros

1. `dword` `Machine`
   Número de máquina de la que se quiere saber si es o no manejada desde el panel.

17

```
bool LedOff(dword LedNumber)
```

Descripción general

Este método permite desactivar un LED determinado de los que se encuentran en las teclas del visualizador.

Retorno

`true`
Si el número de LED indicado es correcto.
`false`
Si el número de LED indicado NO es correcto.

Parámetros

1. `dword` `LedNumber`
   Indica el número de LED que se desea activar. En el visualizador que nos ocupa este parámetro puede tomar valores 0 a 4 para activar los LEDs de las teclas de función F1 a F5 o bien tomar valores 12 y 13 para activar los LEDs correspondientes a las teclas de función S1 y S2.

18

```
bool LedOn(dword LedNumber)
```

Descripción general

Este método permite activar un LED determinado de los que se encuentran en las teclas del visualizador.

Retorno

`true`
Si el número de LED indicado es correcto.
`false`
Si el número de LED indicado NO es correcto.

Parámetros

2. `dword` `LedNumber`
   Indica el número de LED que se desea activar. En el visualizador que nos ocupa este parámetro puede tomar valores 0 a 4 para activar los LEDs de las teclas de función F1 a F5 o bien tomar valores 12 y 13 para activar los LEDs correspondientes a las teclas de función S1 y S2.

19

```
void Load(string File)
```

Descripción general

Este método carga el archivo un archivo de configuración. Solamente se puede mantener cargado e memoria un archivo de configuración con lo cual si se invoca dos veces este método se descargará el primer archivo cargado. Este método está diseñado para ejecutarse desde las funciones de arranque frío o caliente. En ningún caso debería ejecutarse en el código cíclico.
El formato de este archivo se explicará después.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

2. `string` `File`
   Nombre del archivo a cargar.

20

```
dword RecvKeyCode()
```

Descripción general

Este método permite obtener la última tecla pulsada en el visualizador.

Retorno

`dword`
Retorna en el valor numérico de la tecla pulsada (para saber cuales son los valores numéricos de cada una de las teclas debe consultarse la documentación del visualizador). Tiene además la propiedad de que cada vez que se invoca el método y había una tecla pulsada se elimina la misma de los valores memorizados, con lo cual el método también sirve como detector de flanco de pulsación.

Parámetros

Este método no tiene parámetros de entrada.

21

```
bool SendBinMessage(word nNumber,
                    bool hCopy)
```

Descripción general

Este método se establece por compatibilidad con el componente Piltz_305_IBS, y equivale a una llamada a método "SetScreenNumber". El parámetro hcopy se ignora.

Retorno

`true`
Se ha conseguido introducir el comando en la pila.
`false`
NO se ha conseguido introducir el comando en la pila

Parámetros

3. `word` nNumber
   Número del mensaje que se desea visualizar.
4. `bool` hCopy
   Será `true` si se desea copia impresa o `false` si no se desea realizar copia impresa.

22

```
void SetAlarm(word AlarmNumber)
```

Descripción general

Este método establece una alarma en el sistema. Una alarma es un evento de prioridad máxima, cuando un evento de este tipo se produce el visualizador pasa a presentar la información del mismo en la pantalla, es decir el visualizador cambia el mensaje al número de alarma presentado en el parámetro indicado. Cuando este método es invocado el visualizador encola la alarma en una cola de alarmas. Mientras haya algo en la cola de alarmas no se mostrarán más mensajes que los de alarma. Estos mensajes se mostrarán de forma iterativa en el visualizador hasta que desaparezcan. Solamente desaparecen por código mediante la invocación de los métodos correspondientes.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

2. `word` AlarmNumber
   Número de mensaje de alarma a visualizar.

23

```
bool SetScreenNumber(byte Page)
```

Descripción general

Establece el número de página activa del visualizador al nuevo valor.

Retorno

`true`
Se ha conseguido introducir el comando en la pila.
`false`
NO se ha conseguido introducir el comando en la pila

Parámetros

1. `byte` `Page`

Nueva página a mostrar.

24

```
void SetTransportAlarm(dword Offset,
                       dword Message,
                       dword ConveyorNumber)
```

Descripción general

Este método establece una alarma de transportador. Estas alarmas funcionan como las alarmas normales o convencionales pero están especialmente preparadas para su uso con transportadores.

Retorno

`void`

Este método no retorna nada.

Parámetros

4. `dword` `Offset`

Mensaje a partir del cual se sitúan los mensajes.

5. `dword` `Message`

Número de mensaje a partir del offset establecido.

6. `dword` `ConveyorNumber`

Número de transportador que genera la avería.

25

```
void SetVarForAllMachines(string Variable,
                          dword Value)
```

Descripción general

Este método sirve para comandar simultáneamente el cambio de una variable para todas las máquinas configuradas como controladas desde este visualizador. Por ejemplo su utilización es para eliminar el manual simultáneamente en todos los transportadores de la zona controlada por dicho visualizador.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

3. `string` `Variable`

Nombre de la variable que se desea manipular en todas las máquinas

4. `dword` `Value`

Valor que se desea asignar a la variable en todas las máquinas.

26

```
void SetWarning(word WarningNumber)
```

Descripción general

Este método establece un aviso en el sistema. Un aviso es un evento de prioridad media, cuando un evento de este tipo se produce el visualizador pasa a presentar la información del mismo en la pantalla es decir el visualizador cambia el mensaje al

número de aviso presentado en el parámetro indicado si no hay alarmas pendientes. Cuando este método es invocado el visualizador encola los avisos en una cola. Mientras haya algo en la cola de alarmas no se mostrarán más mensajes que los de alarma. Cuando la cola de alarmas esté vacía se mostrarán los avisos. Estos mensajes se mostrarán de forma iterativa en el visualizador hasta que desaparezcan. Solamente desaparecen por código mediante la invocación de los métodos correspondientes.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

2. `word WarningNumber`
   Número de mensaje de Warning a visualizar.

27

```
word TransportAlarmCount()
```

Descripción general

Este método sirve para saber la cantidad de alarmas de transportador insertadas en la cola de alarmas.

Retorno

`word`

Retorna la cantidad de alarmas insertadas en la cola.

Parámetros

Este método no tiene parámetros de entrada.

28

```
word WarningCount()
```

Descripción general

Este método sirve para saber la cantidad de avisos insertados en la cola de avisos.

Retorno

`word`

Retorna la cantidad de avisos insertados en la cola.

Parámetros

Este método no tiene parámetros de entrada.

4.2.2.2 El fichero de configuración de mensajes

El sistema implementado sigue el mismo patrón y funcionamiento que para la configuración del componente PXT_305_Interbus. La única diferencia añadida es la posibilidad de ligar las variables de Galileo directamente con variables de pantalla del terminal. Para realizar este enlazado, en el fichero INI de configuración, las variables que se declaran para cada pantalla deben especificar un parámetro más, llamado "*AddressX*" (donde X es el número de variable de la pantalla).
Ejemplo de archivo de configuración:

```
; REGLAS BASICAS
; Los archivos INI admiten como comentario cualquier linea que
comience con ;el simbolo de punto y coma
; no es conveniente poner un comentario en una linea con datos
; La sección de cabecera contiene los datos generales a todo el
INI
[header]
; El campo mensajes contiene las pantallas asociadas al
visualizador
;PXT_305 que llevan algún tipo de autogestión manual

  mensajes=3,12,13,14,15,16,17,18,19

; El campo maquinas contiene la lista de máquinas que se asocia a
este
; visualizador. El visualizador permitirá avanzar entre ellas
hacia
; delante o hacia atrás

  maquinas=1,2,3,4,5,6,7,8,9,10

; Por cada pantalla que aparezca en la entrada de mensajes debe
aparecer
; aqui una sección con dicho número
; estas definiciones de pantallas permiten realizar las siguientes
;funciones:
; 1°- Asociar Variables a variables internas de la pantalla PILZ a
través de
;     Var(x) y Tipo(x)
; 2°- Asociar funcionamientos seguimientos de variables
automáticos es decir
;     al pulsar cualquier tecla de función ( F1 a F12 )

[3]
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
  Address0=DB2.DBW0
[12]
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
  Address0=DB2.DBW0
  F1      = M_AdelanteManual
  F3      = M_SiguienteManual
  F5      = M_AtrasManual
  VarMaquina = PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
[13]
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
  Address0=DB2.DBW0
  F1      = M_AdelanteManual
  F3      = M_SiguienteManual
  F5      = M_AtrasManual
  VarMaquina = PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
[14]
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
```

```
  Address0=DB2.DBW0
  F1      = M_AdelanteManual
  F3      = M_SiguienteManual
  F5      = M_AtrasManual
  F11     = M_SubirManual
  F12     = M_BajarManual
  VarMaquina = PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
[15]
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
  Address0=DB2.DBW0
  F1      = M_AdelanteManual
  F2      = M_AdelanteManual
  F3      = M_SiguienteManual
  F4      = M_AtrasManual
  F5      = M_AtrasManual
  F11     = M_SubirManual
  F12     = M_BajarManual
  VarMaquina = PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
[16]
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
  Address0=DB2.DBW0
  F1      = M_AdelanteManual
  F2      = M_AntihorarioManual
  F3      = M_SiguienteManual
  F4      = M_HorarioManual
  F5      = M_AtrasManual
  F11     = M_SubirManual
  F12     = M_BajarManual
  VarMaquina = PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
[17]
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
  Address0=DB2.DBW0
  F1      = M_AdelanteManual
  F3      = M_SiguienteManual
  F5      = M_AtrasManual
  F11     = M_AbrirManual
  F12     = M_CerrarManual
  VarMaquina = PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
[18]
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
  Address0=DB2.DBW0
  F1      = M_AdelanteManual
  F2      = M_IzquierdaManual
  F3      = M_SiguienteManual
  F4      = M_DerechaManual
  F5      = M_AtrasManual
  VarMaquina = PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
[19]
  Var0=PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz
  Tipo0=16
  Address0=DB2.DBW0
  F1      = M_AdelanteManual
```

```
  F2      = M_IzquierdaManual
  F3      = M_SiguienteManual
  F4      = M_DerechaManual
  F5      = M_AtrasManual
  F11     = M_AbrirManual
  F12     = M_CerrarManual
  VarMaquina = PUPITRE_ZONA_1.MW_SeleccionMesaManualPilz

; Esta sección define asociaciones entre el valor del TipoManual
de cada
; máquina y cual es la pantalla
; que realizará el manejo de la mima.
; Cada tipo de movimiento se configura mediante un bit:
; Bit0: Adelante Rodillos
; Bit1: Atras    Rodillos
; Bit2: Adelante Cadenas
; Bit3: Atras    Cadenas
; Bit4: Subir
; Bit5: Bajar
; Bit6: Giro     Horario
; Bit7: Giro     Antihorario
; Bit8 a 31: Definidos por usuario
; El Primer número en el TipoManual consiste en el valor decimal
; correspondiente
; Ej: 3 = 12 quiere decir que como 3 en binario son los bit 0 y 1
activos el
; transportador soporta adelante / atras rodillos
; y que en el caso en el que la máquina a controlar sea de este
tipo se
; enviará al usuario a la pantalla 50

[TiposManual]
  1 = 50
  3 = 50
  51 = 52
  63 = 53
  255 = 54
  771 = 55
  3075 = 56
  3843 = 57
```

En todo caso los nombres de variables que se indican deben corresponder al nombre de una de las variables que existen en el programa Xana.

Los nombres de las variables están limitados a 200 caracteres.

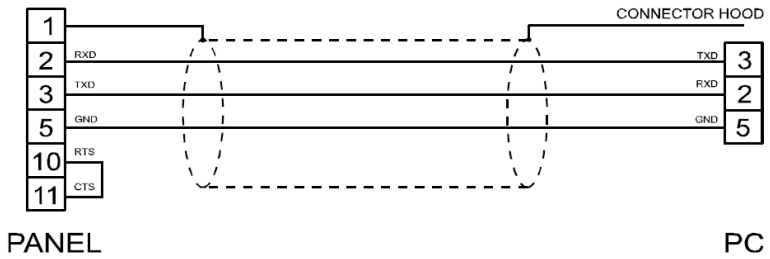
El formato de las direcciones de las variables debe ser el mismo que el usado por la aplicación de configuración de Exor, y todas las variables del la aplicación deben estar mapeándose en el DB2.
Ejemplos:

BIT 5.1 → DB2.DBX5.1
BYTE 2 → DB2.DBB2
WORD 6 → DB2.DBW6

DWORD 10 → DB2.DBD10

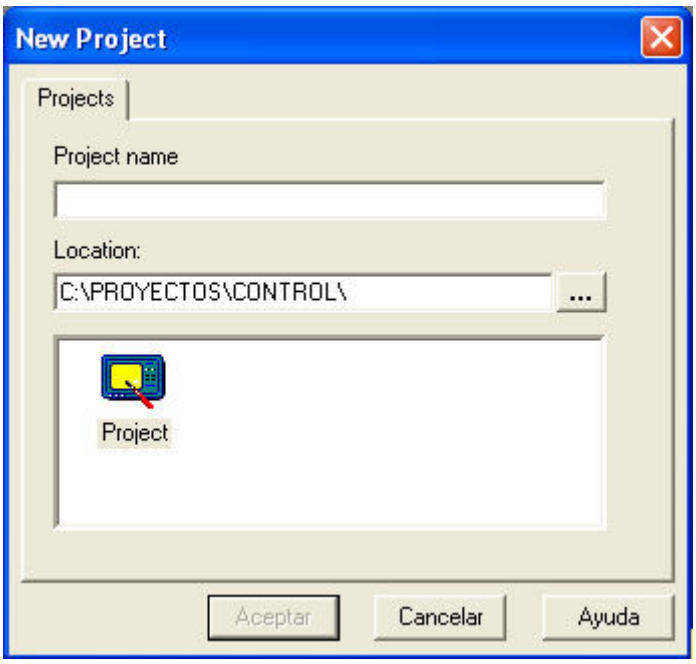
4.2.2.3 Configuración del Visualizador

Para configurar la UniOP, el primer paso es conectar el terminal con un PC de desarrollo donde estará instalada la aplicación "Designer" de Exor. Esta conexión se realiza mediante un cable CA2 del puerto PC/PRINTER PORT del terminal al ordenador principal (puerto COM) desde donde se pretende configurarlo.
El esquema de conexionado puede verse en la siguiente ilustración:

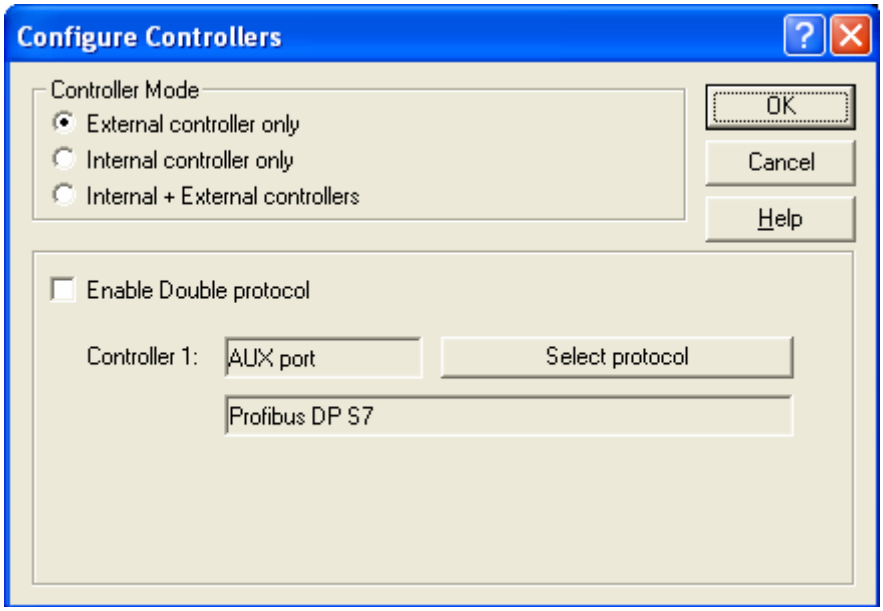


De este panel existen dos versiones. Una tiene 4 líneas de cristal líquido (LC-Display) con 20 caracteres, 12 teclas de función (F1…F12), 23 numéricas y de control y 12 LEDs. La otra versión dispone de una pantalla táctil STN con una resolución de 320x240, y 32 Mb de memoria. La memoria de estos paneles es flash-EPROM.

En cada mensaje o pantalla se pueden insertar variables referidas a una periferia simulada que siempre debe situarse en el ***DB2*** del PLC. El tamaño máximo disponible de esta memoria simulada es de 8192 bytes. En el fichero .ini de configuración se explica como ligar estas direcciones virtuales con variables de Galileo, de forma que se actualicen en ambos sentidos sus valores al ser mostrados en el panel del operador. Los valores de estas variables pueden ser modificados desde el teclado y enviados a Galileo o pueden ser modificados en el Galileo y enviadas al visualizador, sin necesidad de ninguna programación extra por parte de control.



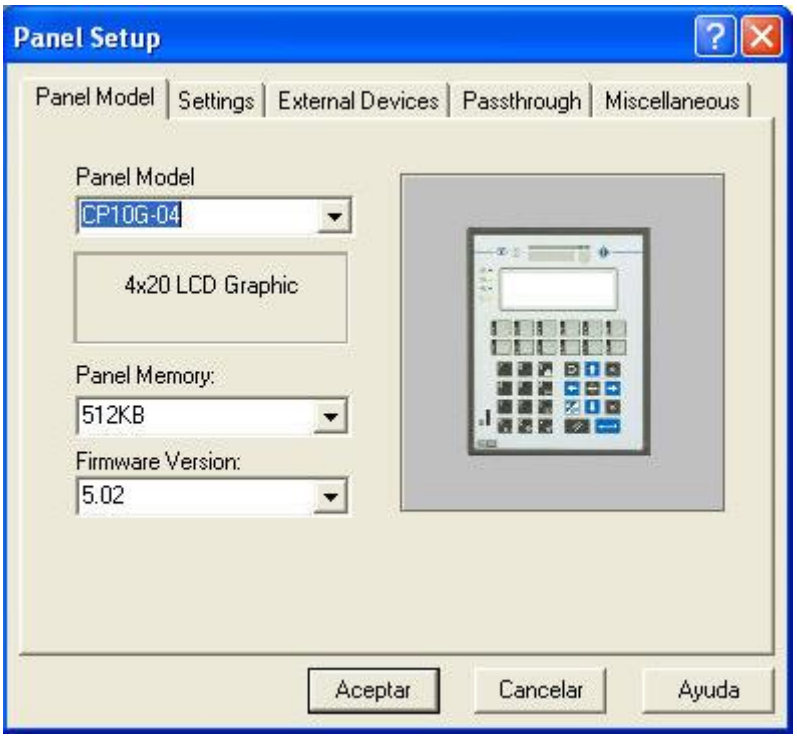
El siguiente paso es configurar el controlador a usar, para eso accedemos a la pantalla siguiente a través del menú "*Project → Configure Controller*"



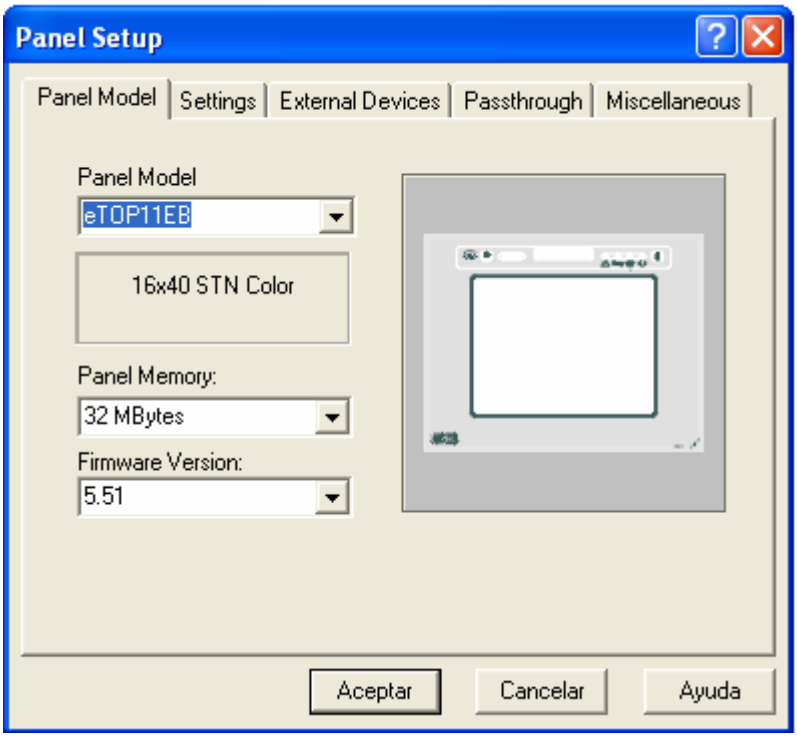
NOTA: Se debe seleccionar como controlador el Profibus DP S7 como controlador externo. El funcionamiento del protocolo del componente depende de este parámetro, y si se usa otro, la comunicación entre la UniOP y el componente fallará.

Seleccione *"Project → Panel Setup"* para seleccionar el tipo de pantalla de display representativo de panel UniOP instalado:

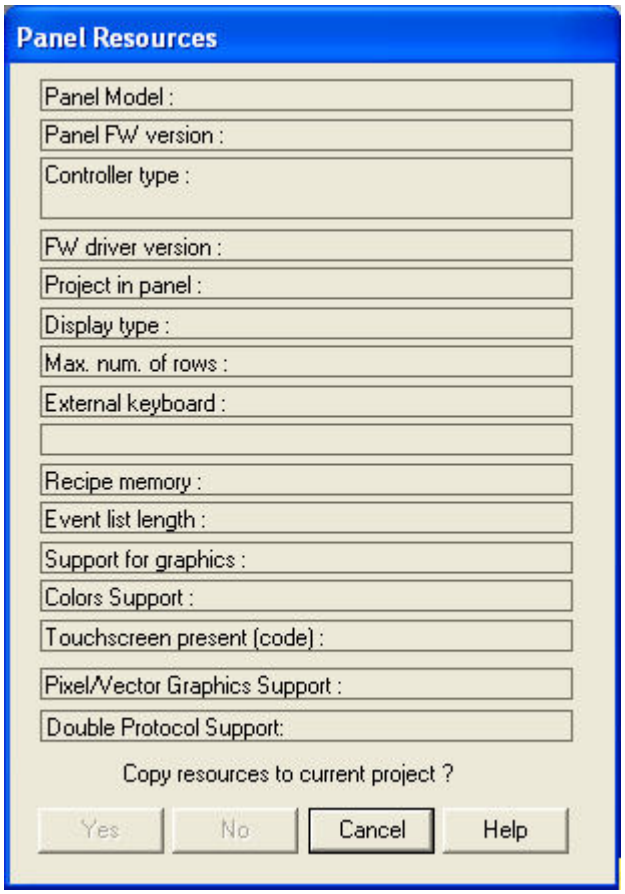
- Para el LCD:



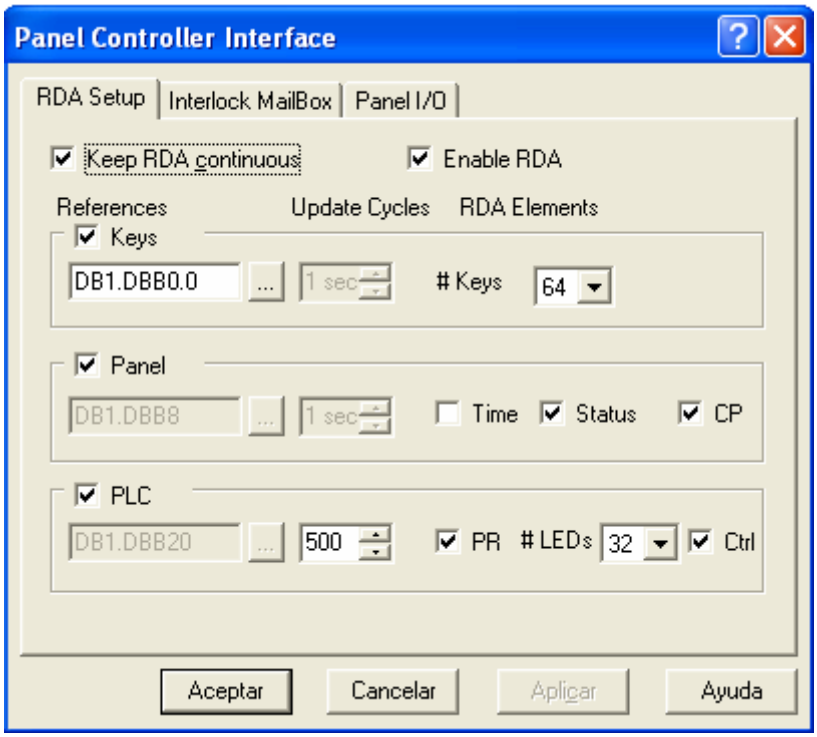
- Para la pantalla táctil:



Este paso puede obviarse si el panel está en modo configuración y utilizando la opción *"Get Panel Resources"* del menú *"Transfers"*. Esto hace que el programa detecte automáticamente y cargue las características del modelo instalado. En ese caso, al usar esa opción, y si la conexión por cable es correcta, se mostrará una pantalla similar a la siguiente, donde una vez rellenos los datos, con sólo pulsar el botón "Yes", se procederá a la auto-configuración del programa.



La última parte de la configuración "estándar" se realiza a través del menú *"Project → Panel Controller Interface"*, lo que nos lleva a la siguiente pantalla:

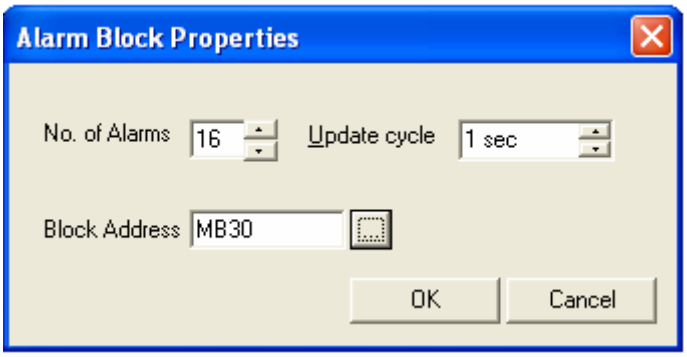


Es importante que configuremos los valores EXACTAMENTE igual que se muestran en esta pantalla. De ello depende que el componente pueda seguir y interactuar con el panel. El resto de los tabs de esta pantalla no debemos tocarlos (por defecto, sus opciones están desactivadas).

En este punto ya estamos listos para diseñar las páginas que deseemos que se muestren desde el panel. Debemos tener presente que la asignación de numero de pantalla es automática por parte del programa, para después plasmar ese número en el fichero de configuración del componente.

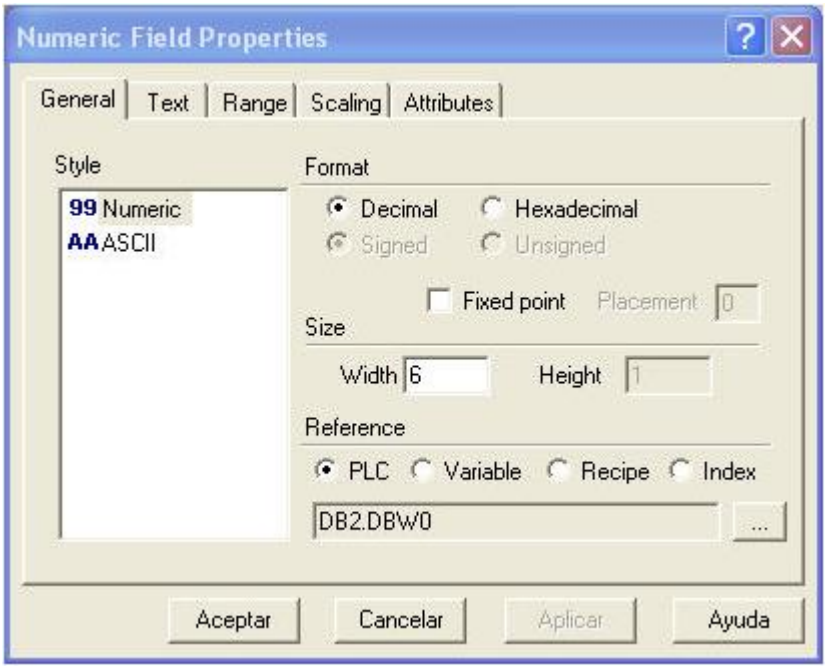
Uno de los puntos principales por configurar es la lista de alarmas que se pueden activar o desactivar desde el componente. Estas se definen como una lista cerrada (de un máximo de 16 elementos) y se pueden editar de la siguiente manera:

1. Es necesario crear un bloque de alarmas. Para ello, se pulsa con el botón derecho en el nodo "Alarms" del árbol del proyecto y se selecciona la opción "Add block".
2. Aparecerá un formulario para crear el bloque. Se tiene que configurar con los siguientes valores:

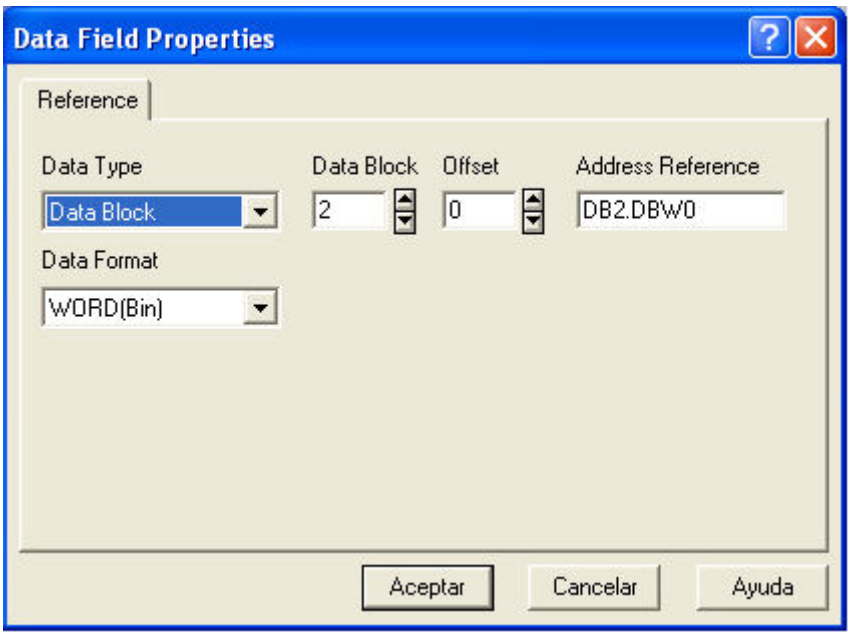


Es importante que no se cambie la configuración del bloque aquí mostrada. Todas las alarmas deben empezar a partir de ese byte del DB1. Asimismo, por cada alarma podemos configurar varias características, aunque quizás la más útil sea la de permitir que cuando se active una alarma, el visualizador cambie a una pantalla predefinida en el proyecto. Esto se realiza haciendo doble clic en el nodo que representa el nuevo bloque de alarmas creada. Aparecerá una pantalla en el que se pueden definir las alarmas y la página a la que se salta cuando se activa.

Además de las alarmas, el otro punto importante son las variables que pretendemos mostrar. Dichas variables van a ser mapeadas en un bloque de memoria ficticio del componente, que siempre se identificará como DB2. Para insertar una referencia a un campo en una pantalla del proyecto, usaremos la opción "*Insert - Data Fields - Numeric/ASCII*". El cursor se transformará en una cruz, que usaremos para definir los límites del campo que pretendemos mostrar. Tras esto, el diálogo de propiedades aparecerá en pantalla:



Pulsando el botón "**...**" de Referencia, podremos establecer la dirección de memoria física que queremos asignarle a esa variable. Esto se hará mediante el diálogo de "Data Field Properties" que se muestra a continuación:

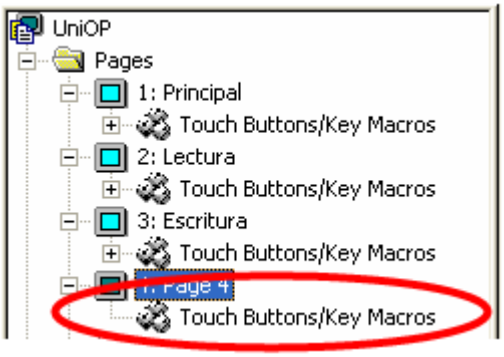


NOTA: Usar siempre el DATA BLOCK 2.

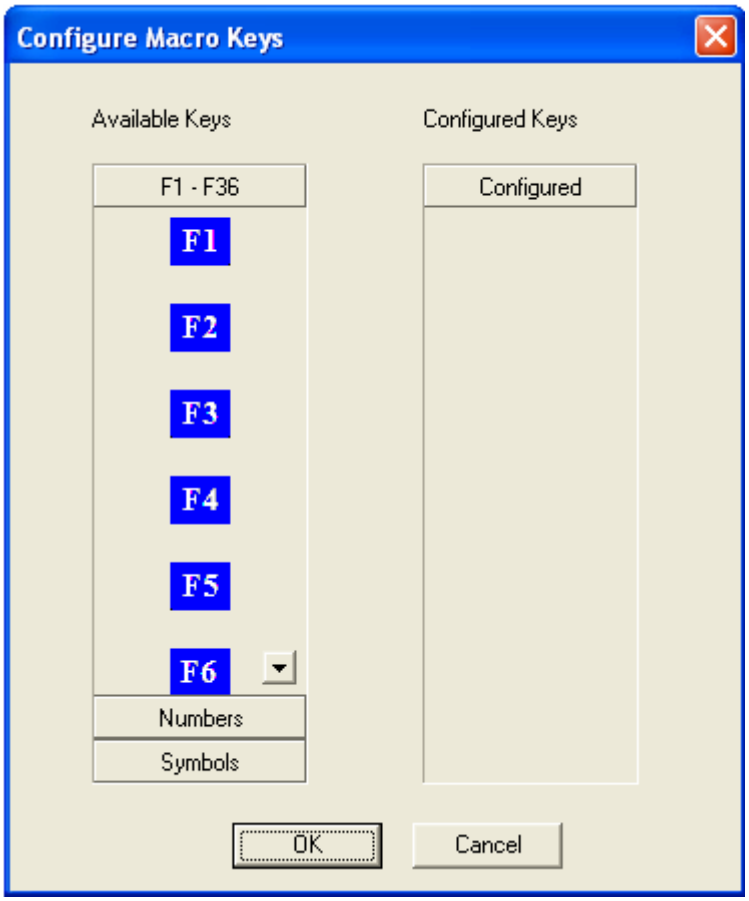
El "Data Format" nos servirá para establecer el tamaño de la variable que queremos mostrar (byte, word, dword). Es importante usar siempre el formato "Bin" y no los otros.

Al establecer el offset, debemos tener en consideración que podemos provocar solapes de la memoria intermedia. Esto puede provocar que visualicemos valores extraños en pantalla cuando carguemos la configuración en la UniOP.

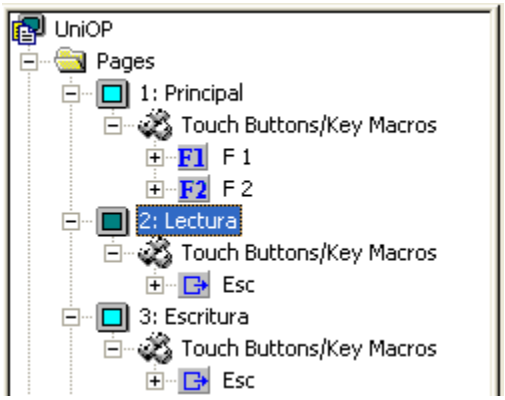
Además de añadir campos numéricos, podemos configurar como queremos que se comporten las pantallas ante la pulsación de las teclas de función o especiales. Para ello basta con desplegar el nodo que representa la pantalla a configurar, hacer clic con el botón derecho en el nodo "*Touch buttons/Key macros*" y seleccionar la opción "Configure Keys":



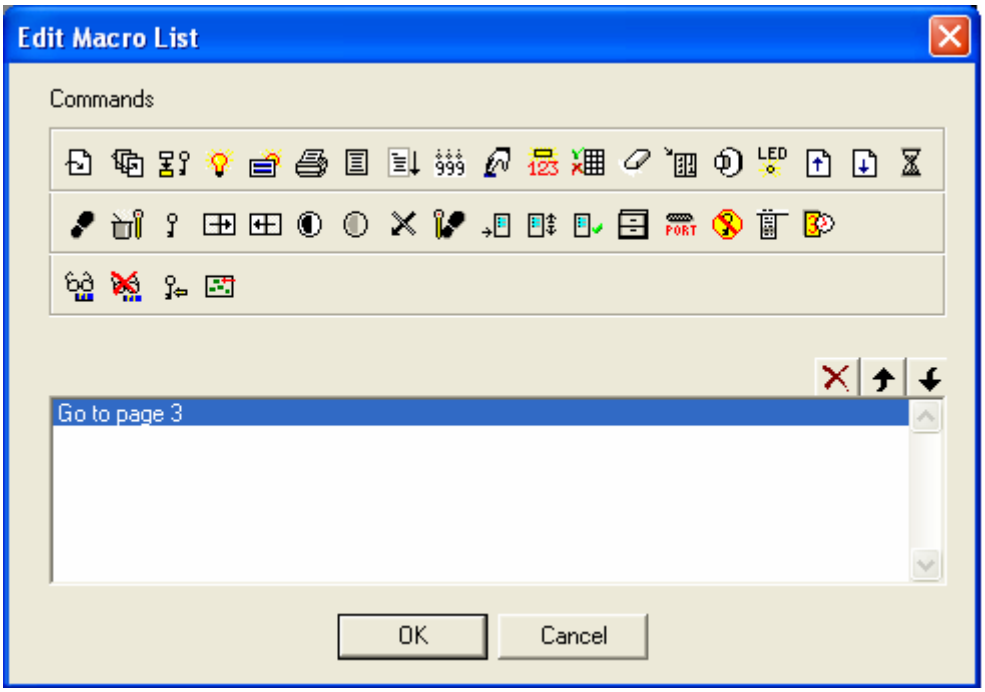
Aparecerá el formulario siguiente:



Desde el que podremos seleccionar las teclas que estarán activas en esta pantalla. Para ello solo es necesario hacer doble clic en la tecla que queramos, de la parte izquierda, y automáticamente pasará a la parte derecha. Una vez hayamos hecho esto, por cada tecla configurada aparecerá un nuevo nodo colgando de "Touch buttons/Key macros", desde el que podremos definir el comportamiento de la pantalla ante su pulsación.

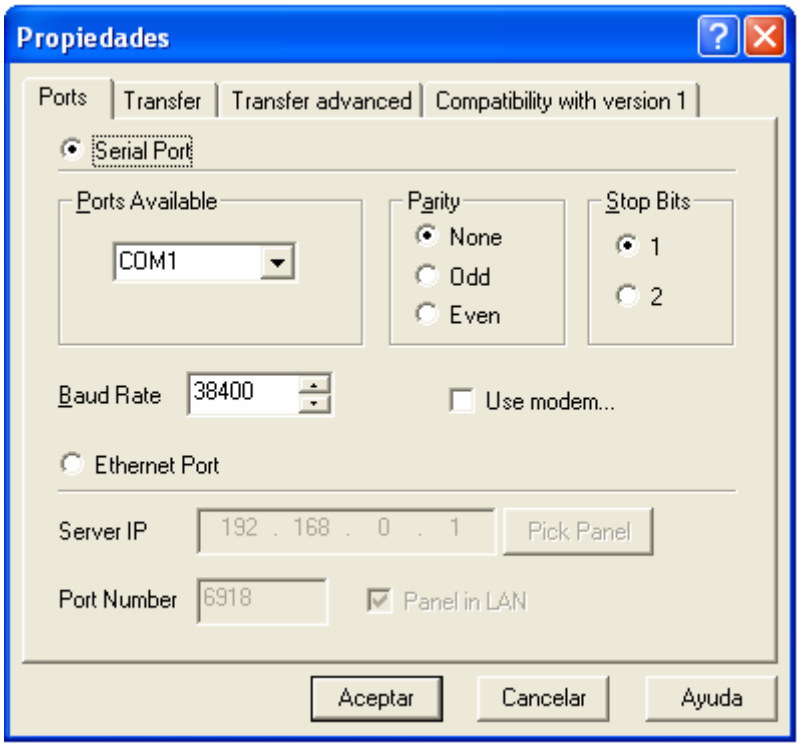


Para cada tecla, aparecerán colgando de su nodo, los posibles eventos a los que responde (pulsación, suelta, etc.), y haciendo doble clic sobre el, podremos definir su comportamiento:



No es objeto de esta guía explicar el comportamiento de este formulario. Como ejemplo, como se configura para que ante la pulsación de la tecla se muestre la pantalla 3.

El último paso, una vez completada la edición de las pantallas que deseemos mostrar, sería transferir la configuración del proyecto al terminal. Para ello usamos la opción "*Transfers → Options → Serial Ports*", desde donde deberemos seleccionar el puerto de comunicaciones que se usará para comunicar con el panel, en el siguiente diálogo:



Los puertos que no estén disponibles estarán desactivados para su selección, En circunstancias normales, use los siguientes parámetros para comunicarse con el panel: 38400 baudios, parity None, y 1 bit de stop.

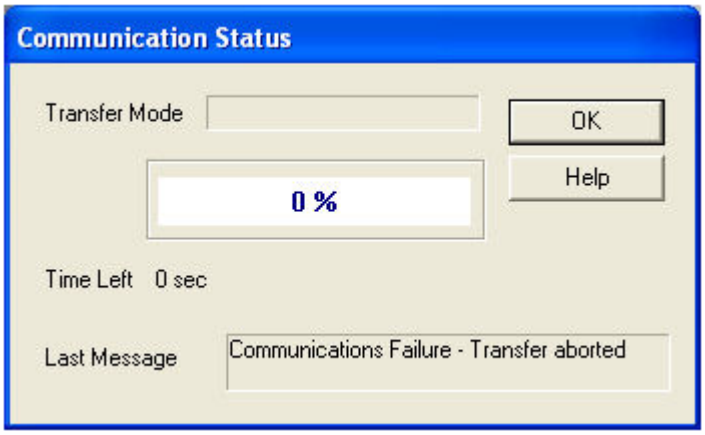
Si el panel UniOP aun no está encendido, enciéndalo en este momento y asegúrese de que está en modo configuración (de otra manera no será posible comunicarse con él). Cuando el panel está en modo configuración, las palabras "CONFIGURATION MODE" aparecen en pantalla. Para entrar en este modo debe:

Presionar la tecla *Enter* o presionar un punto vacío del panel (en el caso de paneles táctiles) hasta que el menú de comandos aparece.

Usando las teclas de dirección, seleccionar *CONFIG o CFG* y presionar *Enter* de nuevo.

Desde el software Designer, seleccionar "*Transfers – Download*" para enviar el proyecto al panel y configurarlo.

En ese momento, si la configuración de puertos es correcta, aparecerá un diálogo como el que se muestra a continuación:



Una vez completada la transferencia, el proyecto comenzará de forma automática, y la luz de "*Fault*" del panel debería estar apagada, mientras que los LEDs de "*Com*" y "*Run*" (si están presentes en el modelo) deberían estar encendidos. Si el LED de "*Com*" parpadea, eso indica un problema de comunicaciones.

4.2.3 ENCODERS_ENCOM

4.2.3.1 Interface

El club Interbus (www.interbusclub.com) ha definido una serie de estándares de comunicaciones para intentar homogeneizar el funcionamiento de protocolos de diálogo con diferentes elementos. Los elementos estandarizados más utilizados por nosotros en nuestro trabajo son Encoders, Variadores de frecuencia y Interfaces Hombre-Máquina. El caso que nos ocupa ahora son los encoders, para trabajar con Interbus se ha definido el estándar ENCOM que engloba varios perfiles de encoders programables.
Se ha desarrollado un componente que realiza el interface con los encoders de tipo ENCOM, básicamente son ENCOM los encoders de Interbus de fabricantes como Fraba, Herrecorr, Hengstler y otros. Los encoders Interbus de T+R con soportan el estándar ENCOM.

NOTA: De todas formas y pese al estándar y al no haber ningún organismo que aprueba la inclusión de un producto en una norma el departamento de desarrollo de Galielo ha observado pequeñas desviaciones de algunos fabricantes respecto a algunos de los comandos del protocolo ENCOM, estos comentarios se realizan en cada una de las funciones de este componente. Se quiere hacer notar que se han probado 3 tipos de encoders que soportan el protocolo ENCOM y que cualquier otro podría sufrir de alguna incompatibilidad similar a las mencionadas.

Adjunta a esta documentación se entrega el estándar ENCOM por si alguien muestra interés en profundizar en el asunto.

A continuación se expone el interface del componente.

Class	`Encoder_ENCOM`
BUS IN	4 bytes

BUS OUT 4 bytes

Listado de métodos:

1	`bool Busy()`
2	`void ClearError()`
3	`dword Error()`
4	`bool Preset()`
5	`bool SetAll(bool CCW, dword Resolution, dword Preset)`
6	`bool SetCCW()`
7	`bool SetCW()`
8	`bool SetPreset(dword Preset)`
9	`bool SetResolution(dword Resolution)`
10	`void StoreHengstler()`
11	`dword Value()`

Descripción de los métodos:

1

```
bool Busy()
```

Descripción general

Retorna un booleano indicando si hay comandos en la pila o no, esto indica que el objeto tiene aún comandos por despachar.

Retorno

`true`
Hay comandos pendientes de despachar en la pila.
`false`
NO hay comandos pendientes de despachar en la pila.

Parámetros

Este método no tiene parámetros de entrada.

2

```
void ClearError()
```

Descripción general

Este método inicia una secuencia de reset del encoder para eliminarle el error. Típicamente esta secuencia de reset produce que una invocación durante el reset al método `Value` devuelva valores absurdos.

Retorno

`void`
Este método no retorna nada.

Parámetros

Este método no tiene parámetros de entrada.

3

```
dword Error()
```

Descripción general

Retorna un `dword` indicando el código de error.

Retorno

`dword`

1. `0x02000000` → Parámetro inválido del Host.
2. `0x04000000` → Número de parámetro desconocido.
3. `0x06000000` → Parámetro perdido.
4. `0x18000000` → Código de error específico del fabricante.
5. `0x1A000000` → Código de error específico del fabricante.
6. `0x1C000000` → Código de error específico del fabricante.
7. `0x1E000000` → Código de error específico del fabricante.
8. `0` → Si no hay error.

Parámetros

Este método no tiene parámetros de entrada.

4

```
bool Preset()
```

Descripción general

Realiza un preset del encoder con el valor del preset configurado previamente mediante `SetPreset` o `SetAll`.

NOTA:

Se ha constatado que los encoders de Herrekor necesitan la llamada a Preset para ejecutar el preset después de un `SetPreset` o un `SetAll`, mientras que los encoders de Hengstler no sólo no lo necesitan, sino que no funcionan si se usa esa llamada.

Retorno

`true`

Si el comando se ha introducido en la pila.

`false`

Si el comando NO ha podido introducirse en la pila.

Parámetros

Este método no tiene parámetros de entrada.

5

```
bool SetAll(bool CCW,
            dword Resolution,
            dword Preset)
```

Descripción general

Este método permite establecer todos los parámetros en una única invocación.

Retorno

`true`
Si el comando se ha introducido en la pila.
`false`
Si el comando NO ha podido introducirse en la pila.

Parámetros

1. `bool` CCW
   Si el valor es true se estará indicando que deseamos configurarlo con cuenta ascendente en sentido de giro contrario a las agujas del reloj, en caso contrario debemos pasar false a este parámetro.
2. `dword` Resolution
   A esta función hay que pasarle un parámetro que corresponde con el número de pasos que se quiere tener por vuelta. El parámetro Resolution admite valores comprendidos entre 1 y 8192 pero teniendo en cuenta que el valor máximo de posición devuelto por el encoder que se va a poder visualizar va a ser 2^{25} (33.554.432) y que para valores de resolución menores de 512 el valor máximo que nos devolverá el encoder, corresponderá a una determinada distancia X y para valores de resolución mayores de 512, el valor máximo que nos devolverá el encoder serán los 2^{25} pero corresponderán a una distancia proporcionalmente inferior a dicha resolución fijada.
3. `dword` Preset
   Podrá tomar cualquier valor, teniendo en cuenta que el valor de la posición actual que el usuario puede ver viene representado en 25 bits.

6

```
bool SetCCW()
```

Descripción general

Establece el sentido de giro ascendente en el sentido contrario al sentido de giro de las agujas del reloj.

Retorno

`true`
Si el comando se ha introducido en la pila.
`false`
Si el comando NO ha podido introducirse en la pila.

Parámetros

Este método no tiene parámetros de entrada.

7

```
bool SetCW()
```

Descripción general

Establece el sentido de giro ascendente en el mismo que el sentido de giro de las agujas del reloj.

Retorno

`true`
Si el comando se ha introducido en la pila.
`false`
Si el comando NO ha podido introducirse en la pila.

Parámetros

Este método no tiene parámetros de entrada.

8

```
bool SetPreset(dword Preset)
```

Descripción general

Esta función permite establecer cual es el valor deseado a la hora de realizar un preset.

Retorno

`true`
Si el comando se ha introducido en la pila.
`false`
Si el comando NO ha podido introducirse en la pila.

Parámetros

1. `dword` `Resolution`
   Podrá tomar cualquier valor, teniendo en cuenta que el valor de la posición actual que el usuario puede ver viene representado en 25 bits.

9

```
bool SetResolution(dword Resolution)
```

Descripción general

Esta función permite configurar la resolución deseada para el encoder.

Retorno

`true`
Si el comando se ha introducido en la pila.
`false`
Si el comando NO ha podido introducirse en la pila.

Parámetros

1. `dword` `Resolution`
   A esta función hay que pasarle un parámetro que corresponde con el número da pasos que se quiere tener por vuelta. El parámetro Resolution admite valores comprendidos entre 1 y 8192 pero teniendo en cuenta que el valor máximo de posición devuelto por el encoder que se va a poder visualizar va a ser 2^{25} (33.554.432) y que para valores de resolución menores de 512 el valor máximo que nos devolverá el encoder, corresponderá a una determinada distancia X y para valores de resolución mayores de 512, el valor máximo que nos devolverá el encoder serán los 2^{25} pero corresponderán a una distancia proporcionalmente inferior a dicha resolución fijada.

10

```
void StoreHengstler()
```

Descripción general

Este método permite salvar los valores almacenados en memoria RAM a la memoria EEPROM en el caso de los encoders de la marca HENGSTLER. Dado que por defecto los encoders de esta marca no almacenan en dicha memoria su configuración tras un cambio de parámetros, si el encoder pierde la tensión de alimentación, se reiniciara a los parámetros salvados en la EEPROM.

NOTA:
Este método está pensado para trabajar única y exclusivamente con los encoders de Hengstler, su invocación cuando se este usando cualquier otro tipo de encoder puede suponer un malfuncionamiento del mismo a corto o largo plazo.

Retorno

`void`

Este método no retorna nada.

Parámetros

Este método no tiene parámetros de entrada.

11

```
dword Value()
```

Descripción general

Este método indica la lectura actual del encoder. Se debe tener especial cuidado al leer el valor del encoder por que cuando el encoder está sufriendo una parametrización la lectura puede ser absurda.

Retorno

`dword`

Retorna el valor de la lectura del encoder.

Parámetros

Este método no tiene parámetros de entrada.

4.2.4 ENCODER T+R

4.2.4.1 Interface

El fabricante T+R suministra un encoder compatible con Interbus-S, este encoder tiene la pega de que no es compatible con el estándar ENCOM, de manera que se ha creado otro componente para realizar el interface con este fabricante de encoders.

La principal ventaja de este modelo de encoder es la elevadísima cantidad de preset que admite el mismo. Cosa que por ejemplo el encoder de Fraba (ENCOM) no soporta.

El nombre del componente a instanciar es *TR_Interbus*. También existe una versión para el encoder TR_KE_65_M, un encoder que funciona mejor en condiciones de frío extremo (almacenes de congelados, por ejemplo) cuya referencia como nombre de objeto es *TR_KE_65_M,* e idéntico comportamiento a nivel de interfaz.

El interface utilizado por este componente es el siguiente:

Class	`TR_Interbus | TR_KE_65_M`
BUS IN	4 bytes
BUS OUT	4 bytes

Listado de métodos:

1	`bool Busy()`
2	`void ClearError()`
3	`bool Error()`
4	`bool Preset(dword Preset)`
5	`bool SetAll(bool CCW, dword Resolution, dword Preset)`
6	`bool SetCCW()`
7	`bool SetCW()`
8	`bool SetResolution(dword Resolution)`
9	`dword Value()`

Descripción de los métodos:

1

```
bool Busy()
```

Descripción general

Retorna un booleano indicando si hay comandos en la pila o no, esto indica que el objeto tiene aún comandos por despachar.

Retorno

`true`
Hay comandos pendientes de despachar en la pila.
`false`
NO hay comandos pendientes de despachar en la pila.

Parámetros

Este método no tiene parámetros de entrada.

2

```
void ClearError()
```

Descripción general

Este método inicia una secuencia de reset del encoder para eliminarle el error. Típicamente esta secuencia de reset produce que una invocación durante el reset al método `Value` devuelva valores absurdos.

Retorno

`void`
Este método no retorna nada.

Parámetros

Este método no tiene parámetros de entrada.

3

```
bool Error()
```

Descripción general

Retorna si el encoder está en error o no.

Retorno

`true`
El encoder se encuentra en error.
`false`
NO hay error en el encoder.

Parámetros

Este método no tiene parámetros de entrada.

4

```
bool Preset(dword Preset)
```

Descripción general

Realiza un preset del encoder al valor deseado.

Retorno

`true`
Retorna este valor si el comando ha sido introducido en la pila.
`false`
Retorna este valor si el comando NO ha sido introducido en la pila.

Parámetros

1. `dword` `Preset`
   Indica el valor al que se desea realizar el preset del encoder.

5

```
bool SetAll(bool CCW,
            dword Resolution,
            dword Preset)
```

Descripción general

Este método permite establecer todos los parámetros en una única invocación.
NOTA:
Este modelo requiere ser reiniciado (quitándole tensión) tras un cambio en la configuración del sentido de giro.

Retorno

`true`
Retorna este valor si el comando ha sido introducido en la pila.
`false`
Retorna este valor si el comando NO ha sido introducido en la pila.

Parámetros

1. `bool` `CCW`
   Si el valor es true se estará indicando que deseamos configurarlo con cuenta ascendente en sentido de giro contrario a las agujas del reloj, en caso contrario debemos pasar false a este parámetro.
2. `dword` `Resolution`
   Valor de resolución a fijar, debe ser `0xFFFFFF` para este modelo (potencia de 2 en esta familia de encoders habitualmente).
3. `dword` `Preset`

Podrá tomar cualquier valor, teniendo en cuenta que el valor de la posición actual que el usuario puede ver viene representado en 25 bits.

6

```
bool SetCCW()
```

Descripción general

Establece el sentido de giro ascendente en el sentido contrario al sentido de giro de las agujas del reloj.
NOTA:
Este modelo requiere ser reiniciado (quitándole tensión) tras un cambio en la configuración del sentido de giro.

Retorno

`true`
Retorna este valor si el comando ha sido introducido en la pila.
`false`
Retorna este valor si el comando NO ha sido introducido en la pila.

Parámetros

Este método no tiene parámetros de entrada.

7

```
bool SetCW()
```

Descripción general

Establece el sentido de giro ascendente en el mismo que el sentido de giro de las agujas del reloj.
NOTA:
Este modelo requiere ser reiniciado (quitándole tensión) tras un cambio en la configuración del sentido de giro.

Retorno

`true`
Retorna este valor si el comando ha sido introducido en la pila.
`false`
Retorna este valor si el comando NO ha sido introducido en la pila.

Parámetros

Este método no tiene parámetros de entrada.

8

```
bool SetResolution(dword Resolution)
```

Descripción general

Esta función permite configurar la resolución deseada para el encoder.

Retorno

`true`
Retorna este valor si el comando ha sido introducido en la pila.
`false`
Retorna este valor si el comando NO ha sido introducido en la pila.

Parámetros

1. dword Preset
Valor de resolución a fijar, debe ser 0xFFFFFF para este modelo (potencia de 2 en esta familia de encoders habitualmente).

9

```
dword Value()
```

Descripción general

Este método indica la lectura actual del encoder. Se debe tener especial cuidado al leer el valor del encoder por que cuando el encoder está sufriendo una parametrización la lectura puede ser absurda.
NOTA:
Se ha constatado que los encoders de Herrekor necesitan la llamada a Preset para ejecutar el preset después de un SetPreset o un SetAll, mientras que los encoders de Hengstler no lo necesitan.

Retorno

dword
Retorna el valor de la lectura del encoder.

Parámetros

Este método no tiene parámetros de entrada.

4.2.5 ENCODER PB CLASE 2

4.2.5.1 Interface

La clase 2 de Profibus sirve para definir un estándar de encoders que funcionan en este bus de campo con unas características determinadas. Este estándar permitiría manejar cualquier encoder que lo soportase mediante este componente. El fabricante T+R, por ejemplo, suministra un encoder compatible con Profibus-DP.

Este componente es mucho más sencillo de manejar desde el componente que los de Interbus, dado que la configuración sobre su resolución y sentido de giro sólo se hace una única vez desde la aplicación "Sycon".

El nombre del componente a instanciar es:

Class	EncoderPBClase2
BUS IN	4 bytes
BUS OUT	4 bytes

Listado de métodos:

1	bool Busy()
2	bool Preset(dword Preset)
3	dword Value()

Descripción de los métodos:

1

```
bool Busy()
```

Descripción general

Retorna un booleano indicando si hay comandos en la pila o no, esto indica que el objeto tiene aún comandos por despachar.

Retorno

`true`
Hay comandos pendientes de despachar en la pila.
`false`
NO hay comandos pendientes de despachar en la pila.

Parámetros

Este método no tiene parámetros de entrada.

2

```
bool Preset(dword Preset)
```

Descripción general

Realiza un preset del encoder al valor deseado.

Retorno

`true`
Retorna este valor si el comando ha sido introducido en la pila.
`false`
Retorna este valor si el comando NO ha sido introducido en la pila.

Parámetros

1. `dword Preset`
   Indica el valor al que se desea realizar el preset del encoder.

3

```
dword Value()
```

Descripción general

Este método indica la lectura actual del encoder. Se debe tener especial cuidado al leer el valor del encoder por que cuando el encoder está sufriendo una parametrización la lectura puede ser absurda.

Retorno

`dword`
Retorna el valor de la lectura del encoder.

Parámetros

Este método no tiene parámetros de entrada.

4.2.6 ENCODERS CANOPEN CLASE 2

4.2.6.1 Interface

La clase 2 de CanOpen sirve para definir un estándar de Encoders que funcionan en este bus de campo con unas características determinadas. Este estándar permitiría manejar cualquier encoder que lo soportase mediante este componente. El fabricante Kluber, por ejemplo, suministra un encoder compatible con Profibus-DP.
La configuración del encoder se realiza desde la aplicación Sycon.Net, permitiendo definir algunos parámetros como por ejemplo:

- Rango de medición
- Unidades de medición por vuelta
- Limites
- Etc

La configuración de estos parámetros es independiente del fabricante y siempre que se trate de encoders de clase 2 se realiza de la misma manera, esto es, accediendo desde Sycon.Net al parámetro correspondiente y configurando el valor deseado.

Otra posible configuración que se puede hacer se refiere a la imagen de proceso que se publica en el bus. Este punto ya depende del encoder en cuestión, ya que la clase 2 solo define la funcionalidad que se permite, y en caso de CanOpen, los parámetros accesibles, pero no si estos parámetros son o no mapeables en imagen de proceso. Para lograr un componente lo mas genérico e independiente del fabricante posible, se ha decidido reducir el mapeo de imagen de proceso al mínimo, publicando únicamente la posición actual del encoder y accediedo al resto de opciones (como por ejemplo el preset) mediante mensajería.

El interface del componente es el siguiente:

Class	EncoderCanClase2
BUS IN	4 bytes
BUS OUT	0 bytes

Listado de métodos:

```
1  bool Busy()
2  bool Preset(dword Preset)
3  dword Value()
```

Descripción de los métodos:

1

```
bool Busy()
```

Descripción general

Retorna un booleano indicando si hay comandos en la pila o no, esto indica que el objeto tiene aún comandos por despachar.

Retorno

`true`

Hay comandos pendientes de despachar en la pila.

`false`

NO hay comandos pendientes de despachar en la pila.

Parámetros

Este método no tiene parámetros de entrada.

2

```
bool Preset(dword Preset)
```

Descripción general

Realiza un preset del encoder al valor deseado.

Retorno

`true`

Retorna este valor si el comando ha sido introducido en la pila.

`false`

Retorna este valor si el comando NO ha sido introducido en la pila.

Parámetros

1. `dword Preset`
   Indica el valor al que se desea realizar el preset del encoder.

3

```
dword Value()
```

Descripción general

Este método indica la lectura actual del encoder. Se debe tener especial cuidado al leer el valor del encoder por que cuando el encoder está sufriendo una parametrización la lectura puede ser absurda.

Retorno

`dword`

Retorna el valor de la lectura del encoder.

Parámetros

Este método no tiene parámetros de entrada.

4.2.7 SEW: CONTROLADOR DE POSICIÓN IPOS-SEW

4.2.7.1 Interface

Se ha desarrollado también un componente de interface con las tarjetas de posicionamiento de SEW modelo IPOS. Estas tarjetas implementan tanto un módulo de comunicaciones de bus de campo como un sistema posicionador de eje.

El interface del componente es el siguiente:

Class	Ipos_Sew
BUS IN	6 bytes
BUS OUT	6 bytes

Listado de métodos:

1	bool Busy()
2	dword CurrentAcell(dword Numerator, dword Denominator)
3	dword CurrentPosition()
4	dword CurrentSpeed(dword Numerator, dword Denominator)
5	bool Error()
6	dword GetParameter()
7	void GoToPosition(dword Position)
8	bool HasParamError()
9	void Init(dword Id)
10	bool InPosition()
11	bool IsBrakeOn()
12	void Positioning(dword Position)
13	void PresetResolver(dword PresetValue)
14	bool ReadFinished()
15	bool ReadParameter(dword ParamNumber)
16	void Reset()
17	void ResetParamError()
18	void SetSpeed(dword Speed)
19	void SpeedStop()
20	void Stop()
21	void StopPreset()
22	bool WriteFinished()
23	bool WriteParameter(dword ParamNumber, dword ParamValue)

Descripción de métodos:

1

bool **Busy**()

Descripción general

Informa si el componente tiene tareas pendientes de realizar.

Retorno

true
Si aún existen tareas pendientes en la pila.
false

Si NO se está realizando ninguna acción.

Parámetros

Este método no tiene parámetros de entrada.

2

```
dword CurrentAcell(dword Numerator,
                   dword Denominator)
```

Descripción general

Informa de la aceleración actual a la que se mueve la máquina.
Ver método CurrentSpeed()

Retorno

dword
La aceleración actual con la que se mueve la máquina, multiplicada por los factores de conversión, concretamente de la forma [Aceleracion * Numerator / Denominator].

Parámetros

1. dword Numerator

Factor de conversión numerador.

2. dword Denominator

Factor de conversión denominador.

3

```
dword CurrentPosition()
```

Descripción general

Esta función retorna la posición actual de la máquina controlada por la tarjeta IPOS. Dato que el lector (telémetro, encoder) cierra el bucle de posicionamiento de la tarjeta IPOS normalmente el sistema de control desconoce la posición actual si no es por medio de este método.

Retorno

dword
Retorna la posición actual dada por el variador.

Parámetros

Este método no tiene parámetros de entrada.

4

```
dword CurrentSpeed(dword Numerator,
                   dword Denominator)
```

Descripción general

Informa de la velocidad actual a la que se mueve la máquina.
Los parámetros Numerador y Denominador son un factor de conversión para mostrar el valor en el formato deseado.
La velocidad que devuelve, el componente la calcula siempre como unidades de posición / mseg.,
Si por ejemplo el variador entrega la posición en mm. el variador

devolverá una posición en mm/mseg. multiplicado por el factor de conversión que resulta de `Numerador`/`Denominador`.
Por ejemplo, si el cálculo de la velocidad resulta ser de 3, y los parámetros enviados son `Numerador` = 1 y `Denominador` = 1, la velocidad entregada será de 3 mm/mseg, es decir, de 3 m/seg.
Si el cálculo de la velocidad es 3,5 y `Numerador` = 1 y `Denominador` = 1 el resultado será que el componente devuelve 3, sin embargo si `Numerador` = 10 y `Denominador` = 1 el resultado es de 35.

Retorno

`dword`
La velocidad actual a la que se mueve la máquina, multiplicada por los factores de conversión, concretamente de la forma [Velocidad * `Numerator` / `Denominator`].

Parámetros

1. `dword` `Numerator`

Factor de conversión numerador.

2. `dword` `Denominator`

Factor de conversión denominador.

5

```
bool Error()
```

Descripción general

Este método indica si el variador está o no en error. Sirve para identificar los errores que se producen en el variador. Estos errores se rearman mediante el método `Reset`.

Retorno

`true`
Si el variador se encuentra en error.
`false`
Si el variador NO está en error.

Parámetros

Este método no tiene parámetros de entrada.

6

```
dword GetParameter()
```

Descripción general

Informa del valor de un parámetro previamente solicitado mediante una invocación a `ReadParameter`.

Retorno

`dword`
Retorna el valor del parámetro solicitado en una invocación previa a `ReadParameter`. Este método debe de ser invocado sólo después de que la invocación a `ReadParameter` devuelva `true`.

Parámetros

Este método no tiene parámetros de entrada.

7

```
void GoToPosition(dword Position)
```

Descripción general

Da la orden a la tarjeta IPOS de avanzar a una posición determinada. Esta función se debe invocar para el posicionamiento automático, el juego de parámetros que utiliza es el 1.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `dword` `Position`

Posición de destino que se desea enviar al variador.

8

```
bool HasParamError()
```

Descripción general

Informa si existe algún error en los parámetros del variador.

Retorno

`true`

Si el variador informa de algún error en los parámetros

`false`

Si NO existe ningún error en los parámetros.

Parámetros

Este método no tiene parámetros de entrada.

9

```
void Init(dword Id)
```

Descripción general

Carga en el componente un identificador, de manera que todos lo mensajes de traza lo mostrarán. Sirve para identificar distintas instancias del mismo componente.

Retorno

`void`

Esta función no retorna nada.

Parámetros

1. `dword` `Id`

Identificador numérico.

10

```
bool InPosition()
```

Descripción general

Esta función retorna si la máquina ha llegado o no a la posición de destino. Esta función dejará de marcar InPosition aunque lo esté realmente en el momento que se invoque el método `Stop`.

Retorno

`true`
Si la máquina ha llegado a su posición.
`false`
Si la máquina NO ha llegado a su posición.

Parámetros

Este método no tiene parámetros de entrada.

11

```
bool IsBrakeOn()
```

Descripción general

Indica si el freno está o no actuado, se utiliza como seguridad para saber que la máquina ha parado realmente.

Retorno

`true`
Si el freno está activado.
`false`
Si el freno NO está activado.

Parámetros

Este método no tiene parámetros de entrada.

12

```
void Positioning(dword Position)
```

Descripción general

Da la orden a la tarjeta IPOS de avanzar a una posición determinada. Esta función se debe utilizar para el posicionamiento en manual, el juego de parámetros que utiliza es el 1.

Retorno

`void`
Este método no retorna.

Parámetros

1. `dword Position`

Posición de destino que se desea enviar al variador.

13

```
void PresetResolver(dword PresetValue)
```

Descripción general

Este método presetea el resolver (sólo tiene utilidad si el bucle de posicionamiento se cierra con el resolver). Sirve como preset ya que el valor del resolver puede perderse en muchas situaciones (pérdidas de tensión, caídas de bus, deslizamiento de ruedas).

Retorno

void
Este método no retorna nada.

Parámetros

1. dword Preset
Valor que se establecerá como actual al resolver.

14

```
bool ReadFinished()
```

Descripción general

Informa si la lectura de un parámetro ha terminado.

Retorno

true
Si la operación se ha completado ya con éxito.
false
Si aún quedan ciclos pendientes para la finalización de la operación de lectura del parámetro.

Parámetros

Este método no tiene parámetros de entrada.

15

```
bool ReadParameter(dword ParamNumber)
```

Descripción general

Solicita la lectura de un parámetro del variador.

Retorno

true
Si se pudo leer con éxito el valor en el parámetro.
false
Si NO se pudo leer con éxito el valor en el parámetro.

Parámetros

1. dword ParamNumber
Número de parámetro que se desea escribir.

16

```
void Reset()
```

Descripción general

Resetea el variador, se puede utilizar a modo de rearme.

Retorno

void
Este método no retorna.

Parámetros

Este método no tiene parámetros de entrada.

17

```
void ResetParamError()
```

Descripción general

Reinicia el estado de "Error en parámetros" del variador.

Retorno

`void`

Este método no retorna nada. Simplemente reinicia el estado interno del componente de forma que ya no indica error en los parámetros.

Parámetros

Este método no tiene parámetros de entrada.

18

```
void SetSpeed(dword Speed)
```

Descripción general

Este método coloca el variador en modo SERVO. El juego de rampas utilizadas será el 2.

Retorno

`void`

Este método no retorna.

Parámetros

1. `dword` Speed

Velocidad en rpm a la que se desea hacer avanzar la máquina.

19

```
void SpeedStop()
```

Descripción general

Este método detiene la máquina de la misma manera que el método `Stop`, pero está especialmente indicado para el caso en que se encuentre en modo SERVO.

Retorno

`void`

Este método no retorna nada.

Parámetros

Este método no tiene parámetros de entrada.

20

```
void Stop()
```

Descripción general

Esta función detiene el avance del motor, deshabilita el variador y se da orden de paro.

Retorno

`void`

Este método no retorna.

Parámetros

Este método no tiene parámetros de entrada.

21

```
void StopPreset()
```

Descripción general

Interrumpe el comando de preset de resolver.

Retorno

`void`

Este método no retorna nada.

Parámetros

Este método no tiene parámetros de entrada.

22

```
bool WriteFinished()
```

Descripción general

Informa si la escritura de un parámetro ha terminado.

Retorno

`true`

Si una operación previa de escritura de parámetros ha terminado completamente.

`false`

Si una operación previa de escritura de parámetros NO ha terminado completamente.

Parámetros

Este método no tiene parámetros de entrada.

23

```
bool WriteParameter(dword ParamNumber,
                    dword ParamValue)
```

Descripción general

Solicita la escritura de un valor en un parámetro del variador.

Retorno

`true`

Si se pudo escribir con éxito el valor en el parámetro.

`false`

Si NO se pudo escribir con éxito el valor en el parámetro.

Parámetros

1. `dword ParamNumber`

Número de parámetro que se desea escribir.

2. `dword ParamValue`

Valor del parámetro que se desea escribir.

4.2.8 SEW: CONTROLADOR EN LAZO ABIERTO MOVITRAC

4.2.8.1 Interface

Este componente tiene métodos de configuración y métodos para funcionamiento cíclico. Los métodos de configuración deberían ser invocados en todo caso en el arranque del secuenciador que lo utilice (frío/caliente). Los métodos de funcionamiento cíclico pueden ser invocados en todo momento.

El interface del componente es el siguiente:

Class	Movitrac
BUS IN	6 bytes
BUS OUT	6 bytes

Listado de métodos:

1	`void ConfigAutomatic(dword &CurrentPos, dword MaxSpeed, dword Interval, dword Invert)`
2	`void ConfigPID(dword K, dword Ci, dword Cd)`
3	`void ConfigScale(dword Num, dword Den)`
4	`void ConfigSpeed(dword MaxSpeed, dword MaxAce, dword Period, dword MinSpeed, dword Factor, dword Num, dword Den)`
5	`bool Error()`
6	`dword GetCurrentSpeed()`
7	`bool InPosition()`
8	`void ParamSelector(dword Param)`
9	`void RampSelector(dword Ramp)`
10	`void Reset()`
11	`void SetSpeed(word Speed)`

Descripción de los métodos:

1

```
void ConfigAutomatic(dword &CurrentPos,
                     dword MaxSpeed,
                     dword Interval,
                     bool Invert)
```

Descripción general

Esta función implementa la configuración para el posicionamiento.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `dword` `CurrentPos`
   Valor por referencia de la variable donde se puede ver la posición actual de la máquina.
2. `dword` `MaxSpeed`
   Se configura la velocidad máxima a la que se desplazará en posicionamiento automático.
3. `dword` `Interval`
   Define el intervalo de centraje para dar por correcta una posición.
4. `bool` `Invert`
   Variable booleana donde se indicará si se debe invertir o no el sentido de la velocidad. Será `false` si una consigna positiva al motor produce un incremento de posición, `true` en caso contrario.

2

```
void ConfigPID(dword K,
               dword Ci,
               dword Cd)
```

Descripción general

Este método establece los parámetros del regulador PID.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `dword` `K`
   Constante proporcional, este valor corresponde al valor de la constante proporcional multiplicado por 1000.
2. `dword` `Ci`
   Constante integral (se ha limitado a un 2% el valor de la constante proporcional en el efecto global para evitar en la medida de lo posible oscilaciones).
3. `dword` `Cd`
   Constante proporcional.

3

```
void ConfigScale(dword Num,
                 dword Den)
```

Descripción general

Establece la relación de escala. La escala o factor multiplicado será Num/Den.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

2. `dword` `Num`
   Numerador del factor de escala.
3. `dword` `Den`
   Denominador del factor de escala.

4

```
void ConfigSpeed(dword MaxSpeed,
                 dword MaxAce,
                 dword Period,
                 dword MinSpeed,
                 dword Factor,
                 dword Num,
                 dword Den)
```

Descripción general

Este método debería ser el primer método invocado si se utiliza esta componente en un secuenciador. Mediante este método se configuran las opciones relativas a las velocidades y factores de conversión del variador.

Retorno

`void`

Esta función no retorna nada.

Parámetros

2. `dword` `MaxSpeed`
   Velocidad máxima a la que avanza la máquina en mm/seg.
3. `dword` `MaxAce`
   Máxima aceleración en mm/seg.
4. `dword` `Period`
   Periodo en milisegundos, este período sería el de oscilación de la máquina (o múltiplo de este) y representa el tiempo de duración de la parte "S" de la rampa.
5. `dword` `MinSpeed`
   Será la velocidad mínima a la que el variador/motor produce avance, es necesaria para poder hacer arrancar la máquina.
6. `dword` `Factor`
   Factor de conversión de la distancia multiplicado por mil (pulsos).
7. `dword` `Num`
   Numerador (véase siguiente parámetro).
8. `dword` `Den`
   Denominador. Mediante la división Num/Den se obtiene el factor de conversión mm/seg → rpm del motor, la equivalencia es tal que el componente establece

en la palabra de control del variador (velocidad * Num/Den)

5

```
bool Error()
```

Descripción general

Este método indica si el variador está o no en error. Sirve para identificar los errores que se producen en el variador. Estos errores se rearman mediante el método `Reset`.

Retorno

`true`
El variador está en error.
`false`
El variador NO está en error.

Parámetros

Este método no requiere parámetros de entrada.

6

```
dword GetCurrentSpeed()
```

Descripción general

Este método devuelve la velocidad actual a la que se está moviendo la máquina.

Retorno

`dword`
Valor de la velocidad actual de la máquina.

Parámetros

Este método no requiere parámetros de entrada.

7

```
bool InPosition()
```

Descripción general

Este método retorna si la máquina ha llegado o no a la posición de destino. Dejará de indicar `InPosition` aunque realmente esté en la posición en cuanto se invoque el método de `Stop.`

Retorno

`true`
La máquina ha llegado a la posición.
`false`
La máquina aún no ha llegado a la posición o bien ya se ha invocado el método `Stop` aunque esté en la posición.

Parámetros

Este método no requiere parámetros de entrada.

8

```
void ParamSelector(dword Ramp)
```

Descripción general

Este método selecciona la rampa con la que va a trabajar el variador (parámetros de tipo de rampa 1 o tipo de rampa 2 en el variador).

Retorno

```
void
```

Este método no retorna ningún valor.

Parámetros

```
1. dword Ramp
```

0 → Tipo de rampa 1.
1 → Tipo de rampa 2

9

```
void RampSelector(dword Ramp)
```

Descripción general

Este método selecciona la rampa con la que va a trabajar el variador (parámetros de tipo de rampa 1 o tipo de rampa 2 en el variador).

Retorno

```
void
```

Este método no retorna ningún valor.

Parámetros

```
1. dword Ramp
```

0 → Tipo de rampa 1.
1 → Tipo de rampa 2

10

```
void Reset()
```

Descripción general

Este método activa el reset interno del variador para poder rearmar el defecto del variador.

Retorno

```
void
```

Este método no retorna nada.

Parámetros

Este método no requiere parámetros de entrada.

11

```
void SetSpeed(word Speed)
```

Descripción general

Este método establece la consigna del variador a una velocidad constante determinada y realiza el movimiento a dicha velocidad.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `word` `Speed`
   Indica la velocidad de consigna a la que se desplazará la máquina. Esta consigna es la que se transfiere directamente al variador con lo cual debe de multiplicarse por el factor de conversión a rpm (tiene valor 5 por defecto según especificación de SEW)

4.2.9 SEW: CONTROLADOR DE POSICIÓN IPOS_SEW_EXCEPTION

4.2.9.1 Interface

Se ha desarrollado también un componente de interface con las tarjetas de posicionamiento de SEW modelo IPOS. Estas tarjetas implementan tanto un módulo de comunicaciones de bus de campo como un sistema posicionador de eje.

El interface con dichas tarjetas se realiza mediante un mecanismo de 2 pasos:

- Un programa descargado en la tarjeta IPOS Sew que hace de mero interface
- El componte Xana que realiza la comunicación con el posicionador.

Para controlar este componente existe un interface denominado `Ipos_Sew_Ex`.

Nota: este componente realiza la misma funcionalidad que el componente `IPOS_SEW`, pero utilizando un distinto mapeo en bus (12 `bytes` en lugar de 6), lo que permite que las operaciones de lectura y escritura de parámetros se puedan realizar en paralelo a operaciones normales de imagen de proceso, sin cancelación de éstas.

El interface de este componente es el siguiente:

Class	`Ipos_Sew_Ex`
BUS IN	12 bytes
BUS OUT	12 bytes

Los métodos expuestos por el interface son los mismos que el componente `Ipos_Sew`, por lo que pueden ser consultados en el apartado anterior dedicado a dicho componente.

Este componente no ha sido implementado a la espera del desarrollo de la programación en IPOS SEW.

4.2.10 LENZE: VARIADOR 9300

4.2.10.1 Interface

Se ha desarrollado un componente que permite la comunicación con el variador Lenze 9300, que es utilizado en modo posicionador para los transelevadores (concretamente su uso es en los de tipo INABOX).

El interface del componente es el siguiente:

Class	Lenze
BUS IN	16 bytes
BUS OUT	16 bytes

Listado de métodos:

1	dword CurrentAcell(dword Numerator, dword Denominator)
2	dword CurrentPosition()
3	dword CurrentSpeed(dword Numerator, dword Denominator)
4	bool Error()
5	void GoToPosition(dword Position)
6	bool InPosition()
7	bool IsBrakeOn()
8	void Positioning(dword Position)
9	void PresetResolver(dword Preset)
10	void ResetCanBus()
11	void SetAutoAccelAndSpeed(byte Speed, byte Accel)
12	void SetSpeed(dword Speed)
13	void SetSpeedPercent(dword Speed, bool Sent)
14	void SpeedStop()
15	void Stop()
16	void StopPreset()

Descripción de los métodos:

1

```
dword CurrentAcell(dword Numerator,
                   dword Denominator)
```

Descripción general

Informa de la aceleración actual a la que se mueve la máquina.

Retorno

dword
Retorna la velocidad actual a la que se mueve la máquina, multiplicada por los factores de conversión, de la siguiente forma:
Aceleración * Numerator/Denominator

Parámetros

1. dword Numerator
   Factor de conversión numerador.
2. dword Denominator
   Factor de conversión denominador.

2

```
dword CurrentPosition()
```

Descripción general

Devuelve la posición actual de la máquina dada por el variador.

Retorno

dword
Retorna el valor de la posición actual.

Parámetros

Este método no requiere parámetros de entrada.

3

```
dword CurrentSpeed(dword Numerator,
                   dword Denominator)
```

Descripción general

Informa de la velocidad actual a la que se mueve la máquina.
Los parámetros Numerador y Denominador son un factor de conversión para mostrar el valor en el formato deseado.
La velocidad que devuelve, el componente la calcula siempre como unidades de posición / mseg.,
Si por ejemplo el variador entrega la posición en mm. el variador devolverá una posición en mm/mseg. multiplicado por el factor de conversión que resulta de Numerador/Denominador.
Por ejemplo, si el cálculo de la velocidad resulta ser de 3, y los parámetros enviados son Numerador = 1 y Denominador = 1, la velocidad entregada será de 3 mm/mseg, es decir, de 3 m/seg.
Si el cálculo de la velocidad es 3,5 y Numerador = 1 y Denominador = 1 el resultado será que el componente devuelve 3, sin embargo si Numerador = 10 y Denominador = 1 el resultado es de 35.

Retorno

dword
Retorna la velocidad actual a la que se mueve la máquina, multiplicada por los factores de conversión, de la siguiente forma:
Velocidad * Numerator/Denominator

Parámetros

1. `dword` `Numerator`
Factor de conversión numerador.
2. `dword` `Denominator`
Factor de conversión denominador.

4

```
bool Error()
```

Descripción general

Este método indica si el variador está o no en error. Sirve para identificar los errores que se producen en el variador. Estos errores se rearman mediante el método `Reset`.

Retorno

`true`
Si el variador se encuentra en error.
`false`
Si el variador NO se encuentra en error.

Parámetros

Este método no requiere parámetros de entrada.

5

```
void GoToPosition(dword Position)
```

Descripción general

Da la orden de posicionamiento en automático.

Retorno

`void`
Esta función no retorna nada.

Parámetros

1. `dword` `Position`
Valor de la posición a la que se desea enviar la máquina.

6

```
bool InPosition()
```

Descripción general

Esta función retorna si la máquina ha llegado o no a la posición de destino. Esta función dejará de marcar `InPosition` aunque lo este realmente en el momento que se invoque el método `Stop`.

Retorno

`true`
Si la máquina ha llegado a la posición a la cual se le ha enviado.
`false`
Si la máquina NO ha llegado a la posición a la cual se le ha enviado.

Parámetros

Este método no requiere parámetros de entrada.

7

```
bool IsBrakeOn()
```

Descripción general

Informa si está activo el freno.

Retorno

`true`
Si el variador ha aplicado el freno.
`false`
Si el variador NO ha aplicado el freno.

Parámetros

Este método no requiere parámetros de entrada.

8

```
void Positioning(dword Position)
```

Descripción general

Da la orden de posicionamiento en manual.

Retorno

`void`
Esta función no retorna nada.

Parámetros

1. `dword Position`
   Valor de la posición a la que se desea enviar la máquina.

9

```
void PresetResolver(dword Preset)
```

Descripción general

Este método presetea el resolver (sólo tiene utilidad si el bucle de posicionamiento se cierra con el resolver). Sirve como preset ya que el valor del resolver puede perderse en muchas situaciones (pérdidas de tensión, caídas de bus, deslizamiento de ruedas).

Retorno

`void`
Este método no retorna nada

Parámetros

1. `dword Preset`
   Valor que se establecerá como actual al resolver.

10

```
void ResetCanBus()
```

Descripción general

Fuerza el rearranque del bus CAN del variador (telémetro) por si se hubiera quedado en un estado preoperacional.

Retorno

`void`

Este método no retorna nada

Parámetros

Este método no requiere parámetros de entrada.

11

```
void SetAutoAccelAndSpeed(byte Speed,
                          byte Accel)
```

Descripción general

Establece la velocidad y aceleración del motor en automatico.
Nota: a diferencia del modo manual, en este caso, el valor de velocidad es un porcentaje de la velocidad máxima posible.

Retorno

`void`

Esta función no retorna nada.

Parámetros

1. `byte Speed`
   Valor de la velocidad. Se toma como valor de velocidad el valor absoluto del parámetro pasado (es un porcentaje sobre el valor configurado en los parámetros internos del variador)
2. `byte Accel`
   Valor de la aceleración. Al igual que la velocidad es un valor sobre el configurado en los parámetros internos del variador.

12

```
void SetSpeed(dword Speed)
```

Descripción general

Establece la velocidad y sentido del motor en manual.

Retorno

`void`

Esta función no retorna nada.

Parámetros

1. `dword Speed`
   Valor de la velocidad. Se toma como valor de velocidad el valor absoluto del parámetro pasado (es un porcentaje sobre el valor configurado en los parámetros internos del variador), y se utiliza el signo para indicar el sentido de giro:
   a. Positivo → Avance
   b. Negativo → Retroceso

13

```
void SetSpeedPercent(dword Speed,
                     bool Sent)
```

Descripción general

Establece la velocidad y sentido del motor en manual. Es similar a SetSpeedPercent, pero establece el sentido a partir de un parámetro indenpendiente.

Retorno

`void`

Esta función no retorna nada.

Parámetros

1. `dword` `Speed`
   Valor de la velocidad. Se toma como valor de velocidad el valor absoluto del parámetro pasado (es un porcentaje sobre el valor configurado en los parámetros internos del variador)
2. `bool` `Sent`
   Sentido de giro:
   a. Positivo → Avance
   b. Negativo → Retroceso

14

```
void SpeedStop()
```

Descripción general

Este método detiene la máquina de la misma manera que el método `Stop`

Retorno

`void`

Este método no retorna nada

Parámetros

Este método no tiene parámetros de entrada.

15

```
void Stop()
```

Descripción general

Esta función retorna si la máquina ha llegado o no a la posición de destino. Esta función dejará de marcar `InPosition` aunque lo este realmente en el momento que se invoque el método `Stop`.

Retorno

`void`

Esta función no retorna nada.

Parámetros

Este método no requiere parámetros de entrada.

16

`void StopPreset()`
Descripción general
Interrumpe el comando de preset de resolver.
Retorno
`void` Este método no retorna nada
Parámetros
Este método no requiere parámetros de entrada.

4.2.11 LENZE: VARIADOR PLC - 9300

4.2.11.1 Interface

Se ha desarrollado un componente que permite la comunicación con el variador PLC-Lenze 9300. Este variador controla tres motores, y mediante los métodos de este componente se pueden controlar de manera independiente. Aunque el componente está preparado para manejar los tres motores, en la actualidad solo se disponen de métodos para dos de ellos y ha sido utilizado en la máquina Clasimat de MECALUX S.A.

El interface del componente es el siguiente:

Class	`LenzeAVB`
BUS IN	24 bytes
BUS OUT	24 bytes

Listado de métodos:

1	`dword CurrentAcellY(dword Numerator, dword Denominator)`
2	`dword CurrentAcellZ(dword Numerator, dword Denominator)`
3	`dword CurrentPositionY()`
4	`dword CurrentPositionZ()`
5	`dword CurrentSpeedY(dword Numerator, dword Denominator)`
6	`dword CurrentSpeedZ(dword Numerator, dword Denominator)`
7	`dword CurrentWeightY()`
8	`bool Error()`
9	`word GetErrorCode()`
10	`void GoToPositionY(dword PositionY)`
11	`void GoToPositionZ(dword Position)`
12	`bool InPositionY()`
13	`bool InPositionZ()`
14	`bool IsBrakeOnY()`
15	`bool IsBrakeOnZ()`

16	`bool isSelectedEngine(byte Engine)`
17	`void PositioningY(dword PositionY)`
18	`void PositioningZ(dword Position)`
19	`void PreparePresetY()`
20	`void PreparePresetZ()`
21	`void PresetResolverY(dword Preset)`
22	`void PresetResolverZ(dword Preset)`
23	`void Reset()`
24	`void ResetCanBus()`
25	`void SelectEngine(byte Engine)`
26	`void SetAutoAccelAndSpeedY(byte SpeedY, byte AccelY)`
27	`void SetAutoAccelAndSpeedZ(byte Speed, byte Accell)`
28	`void SetSpeedPercentY(dword SpeedY, bool Sent)`
29	`void SetSpeedPercentZ(dword Speed, bool Sent)`
30	`void SetSpeedY(dword SpeedY)`
31	`void SetSpeedZ(dword Speed)`
32	`void SpeedStopY()`
33	`void SpeedStopZ()`
34	`void Stop()`
35	`void StopPresetY()`
36	`void StopPresetZ()`
37	`void StopY()`
38	`void StopZ()`

Descripción de los métodos:

1

```
dword CurrentAcellY(dword Numerator,
                    dword Denominator)
```

Descripción general

Informa de la aceleración actual a la que se mueve el motor Y.
* Ver método `CurrentSpeedY()`*

Retorno

`dword`

La aceleración actual a la que se mueve el motor Y, multiplicada por los factores de conversión, concretamente de la forma [Aceleración * `Numerator` / `Denominator`].

Parámetros

1. `dword` `Numerator`
   Factor de conversión numerador.
2. `dword` `Denominator`
   Factor de conversión denominador.

2

```
dword CurrentAcellZ(dword Numerator,
                    dword Denominator)
```

Descripción general

Informa de la aceleración actual del motor del eje Z.
* Ver método `CurrentSpeedY()`*

Retorno

`dword`
La aceleración actual a la que se mueve el motor Z, multiplicada por los factores de conversión, concretamente de la forma [Aceleración * `Numerator` / `Denominator`].

Parámetros

1. `dword` `Numerator`
   Factor de conversión numerador.
2. `dword` `Denominator`
   Factor de conversión denominador.

3

```
dword CurrentPositionY()
```

Descripción general

Devuelve la posición actual del motor Y.

Retorno

`dword`
Retorna la posición actual de la máquina en el eje Y.

Parámetros

Este método no requiere parámetros de entrada.

4

```
dword CurrentPositionZ()
```

Descripción general

Devuelve la posición actual del motor del eje Z.

Retorno

`dword`
Retorna la posición actual del eje Z.

Parámetros

Este método no requiere parámetros de entrada.

5

```
dword CurrentSpeedY(dword Numerator,
                    dword Denominator)
```

Descripción general

Informa de la velocidad actual a la que se mueve el motor Y.

Los parámetros `Numerador` y `Denominador` son un factor de conversión para mostrar el valor en el formato deseado.
La velocidad que devuelve, el componente la calcula siempre como unidades de posición / mseg.,
Si por ejemplo el variador entrega la posición en mm. el variador devolverá una posición en mm/mseg. multiplicado por el factor de conversión que resulta de `Numerador/Denominador`.
Por ejemplo, si el cálculo de la velocidad resulta ser de 3, y los parámetros enviados son `Numerador` = 1 y `Denominador` = 1, la velocidad entregada será de 3 mm/mseg, es decir, de 3 m/seg.
Si el cálculo de la velocidad es 3,5 y `Numerador` = 1 y `Denominador` = 1 el resultado será que el componente devuelve 3, sin embargo si `Numerador` = 10 y `Denominador` = 1 el resultado es de 35.

Retorno

`dword`
La velocidad actual a la que se mueve el motor Y, multiplicada por los factores de conversión, concretamente de la forma [Velocidad * `Numerator` / `Denominator`].

Parámetros

1. `dword` `Numerator`
   Factor de conversión numerador.
2. `dword` `Denominator`
   Factor de conversión denominador.

6

```
dword CurrentSpeedZ(dword Numerator,
                    dword Denominator)
```

Descripción general

Informa de la velocidad actual a la que se mueve el motor del eje Z.
* Ver método `CurrentSpeedY()`*

Retorno

`dword`
La velocidad actual a la que se mueve el motor Z, multiplicada por los factores de conversión, concretamente de la forma [Velocidad * `Numerator` / `Denominator`].

Parámetros

1. `dword` `Numerator`
   Factor de conversión numerador.
2. `dword` `Denominator`
   Factor de conversión denominador.

7

```
dword CurrentWeightY()
```

Descripción general

Informa del peso actual en la bandeja.

Retorno

dword
Retorna el peso de la bandeja.

Parámetros

Este método no requiere parámetros de entrada.

8

```
bool Error()
```

Descripción general

Este método indica si el variador está o no en error. Sirve para identificar los errores que se producen en el variador. Estos errores se rearman mediante el método `Reset`.

Retorno

`true`
El variador se encuentra en error.
`false`
El variador NO se encuentra en error.

Parámetros

Este método no requiere parámetros de entrada.

9

```
word GetErrorCode()
```

Descripción general

Informa del error que tiene el variador.

Retorno

`word`
Retorna el código de error del variador

Parámetros

Este método no requiere parámetros de entrada.

10

```
void GoToPositionY(dword PositionY)
```

Descripción general

Da la orden de posicionamiento en automático al motor Y.

Retorno

`void`
Esta función no retorna nada.

Parámetros

1. `dword PositionY`
   Valor de la posición en el eje Y a la que se desea enviar la máquina.

11

```
void GoToPositionZ(dword Position)
```

Descripción general

Da la orden de posicionamiento en automático al motor Z.

Retorno

`void`
Este método no retorna nada.

Parámetros

1. `dword` `Position`
   Posición a la que se desea enviar la máquina en el eje Z.

12

```
bool InPositionY()
```

Descripción general

Esta función retorna si el motor Y ha llegado o no a la posición de destino. Esta función dejará de marcar `InPosition` aunque lo este realmente en el momento que se invoque el método `Stop`.

Retorno

`true`
Si el variador ha llegado a la posición indicada en el eje Y.
`false`
Si el variador NO ha llegado a la posición indicada en el eje Y.

Parámetros

Este método no requiere parámetros de entrada.

13

```
bool InPositionZ()
```

Descripción general

Esta función retorna si el motor Z ha llegado o no a la posición de destino. Esta función dejará de marcar `InPosition` aunque lo este realmente en el momento que se invoque el método `Stop`.

Retorno

`true`
El motor del eje Z ha llegado a la posición indicada.
`false`
El motor del eje Z NO ha llegado a la posición indicada.

Parámetros

Este método no requiere parámetros de entrada.

14

```
bool IsBrakeOnY()
```

Descripción general

Informa si está activo el freno del motor Y.

Retorno

`true`
Si el variador ha aplicado el freno en el motor del eje Y.
`false`
Si el variador NO ha aplicado el freno en el motor del eje Y.

Parámetros

Este método no requiere parámetros de entrada.

15

```
bool IsBrakeOnZ()
```

Descripción general

Informa si está activo el freno del motor del eje Z.

Retorno

`true`
El motor del eje Z tiene el freno aplicado.
`false`
El motor del eje Z NO tiene el freno aplicado.

Parámetros

Este método no requiere parámetros de entrada.

16

```
bool isSelectedEngine(byte Engine)
```

Descripción general

Informa si está seleccionado el motor que se indica en el parámetro `Engine`.

Retorno

`true`
El motor que se encuentra seleccionado en el variador es el indicado en el parámetro `Engine`.
`false`
El motor que se encuentra seleccionado en el variador NO es el indicado en el parámetro `Engine`.

Parámetros

1. `byte` `Motor`
   Motor que se desea seleccionar con la siguiente codificación:
   a. 0 → Ningún motor seleccionado
   b. 1 → Motor del eje Y
   c. 2 → Motor del eje Z

17

```
void PositioningY(dword PositionY)
```

Descripción general

Da la orden de posicionamiento al motor Y para realizar la sobreelevación de la máquina.

Retorno

`void`

Esta función no retorna nada.

Parámetros

2. `dword` `PositionY`
   Valor de la posición en el eje Y a la que se desea enviar la máquina.

18

```
void PositioningZ(dword Position)
```

Descripción general

Da la orden de posicionamiento al motor del eje Z.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `dword` `Position`
   Posición a la que se desea enviar la máquina en el eje Z.

19

```
void PreparePresetY()
```

Descripción general

Este método se utiliza para inicializar la secuencia de un preset en el motor del eje Y.

Retorno

`void`

Este método no retorna nada.

Parámetros

Este método no requiere parámetros de entrada.

20

```
void PreparePresetZ()
```

Descripción general

Este método se utiliza para inicializar la secuencia de un preset en el motor del eje Z.

Retorno

`void`

Este método no retorna nada.

Parámetros

Este método no requiere parámetros de entrada.

21

```
void PresetResolverY(dword Preset)
```

Descripción general

Este método presetea el resolver del motor Y (sólo tiene utilidad si el bucle de posicionamiento se cierra con el resolver).

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `dword` `Preset`
   Valor que se establecerá como actual al resolver del motor del eje Y.

22

```
void PresetResolverZ(dword Preset)
```

Descripción general

Este método presetea el resolver del motor del eje Z (sólo tiene utilidad si el bucle de posicionamiento se cierra con el resolver).

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `dword` `Preset`
   Valor que se establecerá como actual al resolver del motor del eje Z.

23

```
void Reset()
```

Descripción general

Resetea el variador, se puede utilizar a modo de rearme.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

24

```
void ResetCanBus()
```

Descripción general

Fuerza el rearmado del bus can del variador con el telémetro.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

25

```
void SelectEngine(byte Engine)
```

Descripción general

Selecciona el motor a utilizar según una codificación establecida con Lenze.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `byte` Engine

   Motor que se desea seleccionar con la siguiente codificación:
   a. 0 → Ningún motor seleccionado
   b. 1 → Motor del eje Y
   c. 2 → Motor del eje Z

26

```
void SetAutoAccelAndSpeedY(byte SpeedY,
                           byte AccelY)
```

Descripción general

Establece la velocidad y aceleración del motor Y en automático.
Nota: a diferencia del modo manual, en este caso, el valor de velocidad es un porcentaje de la velocidad máxima posible.

Retorno

`void`

Esta función no retorna nada.

Parámetros

1. `byte` SpeedY

   Valor de la velocidad en el eje Y.

2. `byte` AccelY

   Valor de la aceleración en el eje Y.

27

```
void SetAutoAccelAndSpeedZ(byte Speed,
                           byte Accell)
```

Descripción general

Establece la velocidad y aceleración del motor Z en automático.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `byte` Speed

   Valor de la velocidad.

2. `byte` Accell

   Valor de la aceleración.

28

```
void SetSpeedPercentY(dword SpeedY,
                      bool Sent)
```

Descripción general

Establece la velocidad y sentido del motor Y en manual. Es similar a `SetSpeedPercent`, pero establece el sentido a partir de un parámetro independiente.

Retorno

`void`

Esta función no retorna nada.

Parámetros

1. `dword SpeedY`
   Valor de la velocidad. Como valor se toma el absoluto del parámetro pasado (es un porcentaje sobre el valor configurado dentro de los parámetros internos del variador), y se utiliza el signo como sentido de giro:
2. `bool Sent`
   Sentido de giro
   a. Positivo → Avance
   b. Negativo → Retroceso

29

```
void SetSpeedPercentZ(dword Speed,
                      bool Sent)
```

Descripción general

Establece la velocidad y sentido del motor Z en manual. Es similar a `SetSpeedPercent`, pero establece el sentido a partir de un parámetro independiente.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `dword Speed`
   Valor de la velocidad. Como valor se toma el absoluto del parámetro pasado (corresponde a un porcentaje sobre el valor configurado en los parámetros internos del variador)
2. `bool Sent`
   Sentido de giro:
   a. Positivo → Avance
   b. Negativo → Retroceso

30

```
void SetSpeedY(dword SpeedY)
```

Descripción general

Establece la velocidad y sentido del motor Y en manual.

Retorno

`void`

Esta función no retorna nada.

Parámetros

1. `dword SpeedY`

   Valor de la velocidad. Como valor se toma el absoluto del parámetro pasado (es un porcentaje sobre el valor configurado dentro de los parámetros internos del variador), y se utiliza el signo como sentido de giro:
   a. Positivo → Avance
   b. Negativo → Retroceso

31

```
void SetSpeedZ(dword Speed)
```

Descripción general

Establece la velocidad y sentido del motor Z en manual.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `dword Speed`

   Valor de la velocidad. Como valor se toma el absoluto del parámetro pasado (corresponde a un porcentaje sobre el valor configurado en los parámetros internos del variador), y se utiliza el signo como sentido de giro:
   a. Positivo → Avance
   b. Negativo → Retroceso

32

```
void SpeedStopY()
```

Descripción general

Este método detiene el motor Y de la misma manera que el método `Stop`.

Retorno

`void`

Este método no retorna nada.

Parámetros

Este método no requiere parámetros de entrada.

33

```
void SpeedStopZ()
```

Descripción general

Este método detiene el motor seleccionado con `SpeedStop` de la misma manera que el método `Stop`.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

34

```
void Stop()
```

Descripción general

Detiene el motor seleccionado con `SelectEngine`

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

35

```
void StopPresetY()
```

Descripción general

Interrumpe el comando de preset de resolver del motor Y.

Retorno

`void`

Este método no retorna nada.

Parámetros

Este método no requiere parámetros de entrada.

36

```
void StopPresetZ()
```

Descripción general

Interrumpe el comando de preset de resolver del motor del eje Z.

Retorno

`void`

Este método no retorna nada.

Parámetros

Este método no requiere parámetros de entrada.

37

```
void StopY()
```

Descripción general

Detiene el motor del eje Y.

Retorno

`void`

Esta función no retorna nada.

Parámetros

Este método no requiere parámetros de entrada.

38

```
void StopZ()
```

Descripción general

Detiene el motor del eje Z.

Retorno

```
void
```

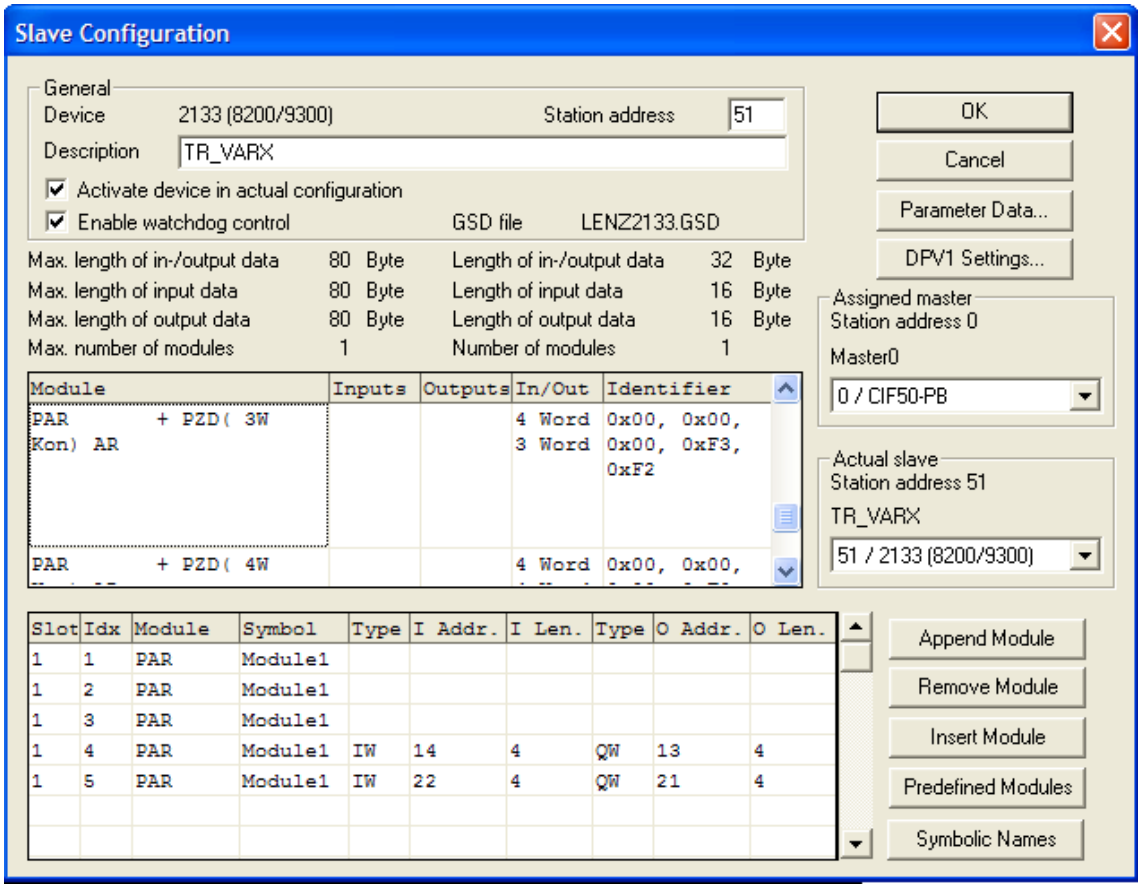
Este método no retorna nada.

Parámetros

Este método no requiere parámetros de entrada.

4.2.11.2 Configuración periferia

La configuración que se ha de asignar a dicho variador para el correcto funcionamiento del componente es la indicada en la imagen:



4.2.12 LENZE: POSICIONADOR ECS

4.2.12.1 Interface

Se ha desarrollado un componente que permite la comunicación con el variador Lenze ECS utilizado habitualmente como posicionador en los transelevadores tipo INABOX.

Class	Lenze
BUS IN	8 bytes
BUS OUT	8 bytes

Listado de métodos:

1	dword CurrentAcell(dword Numerator, dword Denominator)
2	dword CurrentPosition()
3	dword CurrentSpeed(dword Numerator, dword Denominator)
4	bool Error()
5	void GoToPosition(dword Position)
6	bool InPosition()
7	void Positioning(dword Position)
8	void Reset()
9	void SetAutoAccelAndSpeed(word Speed, byte Accel)
10	void SetManualAccel(word Speed, byte ForwardProfile, byte BackwardProfile, byte PositioningProfile)
11	void SetSpeed(dword Speed)
12	void SpeedStop()
13	void Stop()

Descripción de los métodos:

1

```
dword CurrentAcell(dword Numerator,
                   dword Denominator)
```

Descripción general

Informa de la aceleración actual a la que se mueve la máquina.
Ver componente CurrentSpeed()

Retorno

dword
Retorna la velocidad actual a la que se mueve la máquina, multiplicada por los

factores de conversión, de la siguiente forma:
Aceleración * `Numerator/Denominator`

Parámetros

1. `dword` `Numerator`
   Factor de conversión numerador.
2. `dword` `Denominator`
   Factor de conversión denominador.

2

```
dword CurrentPosition()
```

Descripción general

Devuelve la posición actual de la máquina dada por el variador.

Retorno

`dword`
Retorna el valor de la posición actual.

Parámetros

Este método no requiere parámetros de entrada.

3

```
dword CurrentSpeed(dword Numerator,
                   dword Denominator)
```

Descripción general

Informa de la velocidad actual a la que se mueve la máquina.
Los parámetros `Numerador` y `Denominador` son un factor de conversión para mostrar el valor en el formato deseado.
La velocidad que devuelve, el componente la calcula siempre como unidades de posición / mseg.,
Si por ejemplo el variador entrega la posición en mm. el variador devolverá una posición en mm/mseg. multiplicado por el factor de conversión que resulta de `Numerador/Denominador`.
Por ejemplo, si el cálculo de la velocidad resulta ser de 3, y los parámetros enviados son `Numerador` = 1 y `Denominador` = 1, la velocidad entregada será de 3 mm/mseg, es decir, de 3 m/seg.
Si el cálculo de la velocidad es 3,5 y `Numerador` = 1 y `Denominador` = 1 el resultado será que el componente devuelve 3, sin embargo si `Numerador` = 10 y `Denominador` = 1 el resultado es de 35.

Retorno

`dword`
Retorna la velocidad actual a la que se mueve la máquina, multiplicada por los factores de conversión, de la siguiente forma:
Velocidad * `Numerator/Denominator`

Parámetros

1. `dword` `Numerator`
   Factor de conversión numerador.

2. dword Denominator
Factor de conversión denominador.

4

```
bool Error()
```

Descripción general

Este método indica si el variador está o no en error. Sirve para identificar los errores que se producen en el variador. Estos errores se rearman mediante el método Reset.

Retorno

true
Si el variador se encuentra en error.
false
Si el variador NO se encuentra en error.

Parámetros

Este método no requiere parámetros de entrada.

5

```
void GoToPosition(dword Position)
```

Descripción general

Da la orden de posicionamiento en automático.

Retorno

void
Esta función no retorna nada.

Parámetros

1. dword Position
Valor de la posición a la que se desea enviar la máquina.

6

```
bool InPosition()
```

Descripción general

Esta función retorna si la máquina ha llegado o no a la posición de destino. Esta función dejará de marcar InPosition aunque lo este realmente en el momento que se invoque el método Stop.

Retorno

true
Si la máquina ha llegado a la posición a la cual se le ha enviado.
false
Si la máquina NO ha llegado a la posición a la cual se le ha enviado.

Parámetros

Este método no requiere parámetros de entrada.

7

```
void Positioning(dword Position)
```

Descripción general

Da la orden de posicionamiento en manual.

Retorno

`void`

Esta función no retorna nada.

Parámetros

1. `dword` `Position`

   Valor de la posición a la que se desea enviar la máquina.

8

```
void Reset()
```

Descripción general

Fuerza el rearme del variador.

Retorno

`void`

Este método no retorna nada

Parámetros

Este método no requiere parámetros de entrada.

9

```
void SpeedStop()
```

Descripción general

Este método detiene la máquina de la misma manera que el método `Stop`, pero activa el QSP.

Retorno

`void`

Este método no retorna nada

Parámetros

Este método no tiene parámetros de entrada.

10

```
void SetAutoAccelAndSpeed(word Speed,
                          byte Profile)
```

Descripción general

Establece la velocidad y el perfil de funcionamiento del motor en automático.

Retorno

`void`

Esta función no retorna nada.

Parámetros

1. `word` Speed
Valor de la velocidad. Se toma como valor de velocidad el valor absoluto del parámetro pasado (es un porcentaje sobre el valor configurado en los parámetros internos del variador)
2. `byte` Profile
Valor que activa uno de los perfiles definidos en el variador. En total son 16 perfiles definidos mediante cuatro bits de la CW:
0x01 → PNoSet1
0x02 → PNoSet2
0x04 → PNoSet3
0x08 → PNoSet4
De manera estándar se han definido en el variador cuatro perfiles básicos:
- Modo automático → 4
- Modo centraje → 5
- Modo manual adelante → 7
- Modo manual atrás → 8

11

```
void SetManualAccel(word Speed,
                    byte ForwardProfile,
                    byte BackwardProfile,
                    byte PositioningProfile)
```

Descripción general

Establece la velocidad y el perfil de funcionamiento tanto en modo manual como en modo de centraje.

Retorno

`void`
Esta función no retorna nada.

Parámetros

1. `word` Speed
Valor de la velocidad, en el caso de que se realice el movimiento en modo centraje (Positioning()).
2. `byte` ForwardProfile
Valor que activa uno de los perfiles definidos en el variador para el movimiento en modo manual adelante (SetSpeed()).
3. `byte` BackwardProfile
Valor que activa uno de los perfiles definidos en el variador para el movimiento en modo manual atrás (SetSpeed()).
4. `byte` PositioningProfile
Valor que activa uno de los perfiles definidos en el variador para el movimiento en modo centraje (Positioning()).

12

```
void SetSpeed(dword Direction)
```

Descripción general

Realiza el movimiento del motor en modo manual.

Retorno

`void`

Esta función no retorna nada.

Parámetros

1. `dword Direction`

   Sentido de giro del motor en manual. El valor de la velocidad del movimiento en modo manual es fijo y está configurado en el variador.
   - Positivo → Movimiento adelante
   - Negativo → Movimiento atrás

13

```
void Stop()
```

Descripción general

Este método realiza una parada controlada en el variador.

Retorno

`void`

Esta función no retorna nada.

Parámetros

Este método no requiere parámetros de entrada.

4.2.13 LENZE: VARIADOR 9400 L-Force

4.2.13.1 Interface

Se ha desarrollado un componente que permite la comunicación con el variador Lenze L-Force, que se está aplicando por ejemplo en la máquina Clasimat de MECALUX S.A.

El interface del componente es el siguiente:

Class	LenzeLForce
BUS IN	16 bytes
BUS OUT	16 bytes

Listado de métodos:

1	`void BeginWeigh(dword Speed)`
2	`void Config(dword RadioNumerator, dword RadioDenominator,`

	dword RedYNumerator, dword RedYDenominator)
3	dword CurrentAcell(dword Numerator, dword Denominator)
4	dword CurrentPosition()
5	dword CurrentSpeed(dword Numerator, dword Denominator)
6	dword CurrentWeight()
7	bool Error()
8	bool FailureSafetyCh2()
9	bool FailureSafetyCh4()
10	word GetErrorCode(word &Module)
11	void GoToPosition(dword Position)
12	void Init()
13	bool InPosition()
14	bool IsBrakeOn()
15	bool IsControlInhibit()
16	bool IsWatchdogError()
17	bool IsWeighAvailable()
18	void Positioning(dword Position)
19	void PresetResolver(dword Preset)
20	void Reset()
21	void ResetCanBus()
22	void ResetWatchdog()
23	bool SafetyChainError()
24	byte SelectedEngine()
25	void SelectEngine(byte Engine)
26	void SetAutoAccelAndSpeed(byte Speed, byte Accel)
27	void SetManualAccel(byte Speed, byte Accel)
28	void SetSpeed(dword Speed)
29	void SetSpeedPercent(dword Speed, bool Sent)
30	dword SpeedStop()
31	void Stop()
32	void StopPreset()
33	void StopWeight()

Descripción de los métodos:

1

```
void BeginWeigh(dword Speed)
```

Descripción general

Inicia la maniobra de pesaje con la velocidad establecida.

Retorno

void

Este método no retorna nada.

Parámetros

1. dword Speed

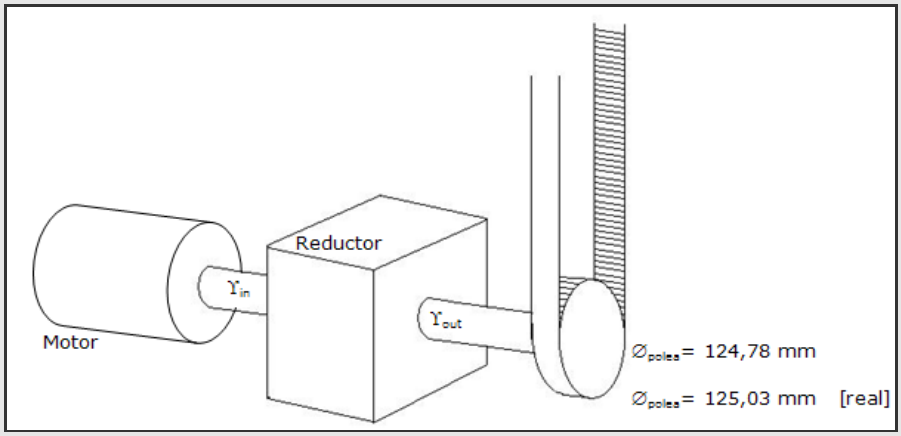
Consigna de velocidad a la que se desea realizar el pesaje.

2

```
void Config(dword RadioNumerator,
            dword RadioDenominator,
            dword RedYNumerator,
            dword RedYDenominator)
```

Descripción general

Establece los valores de radio y reducción en Y a utilizar en el cálculo del peso.



Fórmulas para el cálculo del peso:

$i_{reductor} \cdot \Gamma_{in} = \Gamma_{out}$

$$\Gamma_{out} = F \cdot r_{polea} = m_{cuna} \cdot g \cdot r_{polea} \qquad [N \cdot m]$$

$$m_{cuna} = \frac{i_{reductor} \cdot \Gamma_{in}}{g \cdot r_{polea}} \qquad [kg]$$

Retorno

void

Este método no retorna nada.

Parámetros

1. dword RadioNumerator

Numerador del radio.

2. dword RadioDenominator

Denominador del radio.

3. dword RedYNumerator

Numerador de la reducción en Y.

4. dword RedYDenominator

Denominador de la reducción en Y.

3

```
dword CurrentAcell(dword Numerator,
                   dword Denominator)
```

Descripción general

Informa de la aceleración actual a la que se mueve la máquina.
Ver método CurrentSpeed()

Retorno

dword

Retorna la velocidad actual a la que se mueve la máquina, multiplicada por los factores de conversión, de la siguiente forma:
Aceleración * Numerator/Denominator

Parámetros

1. dword Numerator

Factor de conversión numerador.

2. dword Denominator

Factor de conversión denominador.

4

```
dword CurrentPosition()
```

Descripción general

Devuelve la posición actual de la máquina dada por el variador.

Retorno

dword

Retorna el valor de la posición actual.

Parámetros

Este método no requiere parámetros de entrada.

5

```
dword CurrentSpeed(dword Numerator,
                   dword Denominator)
```

Descripción general

Informa de la velocidad actual a la que se mueve la máquina.
Los parámetros Numerador y Denominador son un factor de conversión para

mostrar el valor en el formato deseado.
La velocidad que devuelve, el componente la calcula siempre como unidades de posición / mseg.,
Si por ejemplo el variador entrega la posición en mm. el variador devolverá una posición en mm/mseg. multiplicado por el factor de conversión que resulta de `Numerador`/`Denominador`.
Por ejemplo, si el cálculo de la velocidad resulta ser de 3, y los parámetros enviados son `Numerador` = 1 y `Denominador` = 1, la velocidad entregada será de 3 mm/mseg, es decir, de 3 m/seg.
Si el cálculo de la velocidad es 3,5 y `Numerador` = 1 y `Denominador` = 1 el resultado será que el componente devuelve 3, sin embargo si `Numerador` = 10 y `Denominador` = 1 el resultado es de 35.

Retorno

`dword`
Retorna la velocidad actual a la que se mueve la máquina, multiplicada por los factores de conversión, de la siguiente forma:
Velocidad * `Numerator`/`Denominator`

Parámetros

1. `dword` `Numerator`

   Factor de conversión numerador.

2. `dword` `Denominator`

   Factor de conversión denominador.

6

```
dword CurrentWeight()
```

Descripción general

Devuelve el peso obtenido durante la maniobra de pesaje.

Retorno

`dword`
Valor del peso que se ha evaluado.

Parámetros

Este método no requiere parámetros de entrada.

7

```
bool Error()
```

Descripción general

Este método indica si el variador está o no en error. Sirve para identificar los errores que se producen en el variador. Estos errores se rearman mediante el método `Reset`.

Retorno

`true`

Si el variador se encuentra en error.
`false`
Si el variador NO se encuentra en error.

Parámetros

Este método no requiere parámetros de entrada.

8

```
bool FailureSafetyCh2()
```

Descripción general

Evalúe el estado del canal 2 de seguridades del módulo del variador.

Retorno

`true`
Existe un error en el canal 2 de la cadena de seguridades. Alguna de ellas se ha accionado.
`false`
El canal 2 de seguridades del módulo del variador está correcto.

Parámetros

Este método no requiere parámetros de entrada.

9

```
bool FailureSafetyCh4()
```

Descripción general

Evalúe el estado del canal 4 de seguridades del módulo del variador.

Retorno

`true`
Existe un error en el canal 4 de la cadena de seguridades. Alguna de ellas se ha accionado.
`false`
El canal 4 de seguridades del módulo del variador está correcto.

Parámetros

Este método no requiere parámetros de entrada.

10

```
word GetErrorCode(word &Module)
```

Descripción general

Devuelve el código de error del variador.

Retorno

`word`
Retorna el código de error del variador.

Parámetros

1. `word` Module

Parámetro por referencia que indica el módulo del variador donde se ha producido el error.

11

```
void GoToPosition(dword Position)
```

Descripción general

Da la orden de posicionamiento en automático.

Retorno

`void`

Esta función no retorna nada.

Parámetros

1. `dword` Position

   Valor de la posición a la que se desea enviar la máquina.

12

```
void Init()
```

Descripción general

Inicial el funcionamiento del componente con el sistema de interpolación de pesos.

Retorno

`void`

Este método no devuelve ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

13

```
bool InPosition()
```

Descripción general

Esta función retorna si la máquina ha llegado o no a la posición de destino. Esta función dejará de marcar `InPosition` aunque lo este realmente en el momento que se invoque el método `Stop`.

Retorno

`true`

Si la máquina ha llegado a la posición a la cual se le ha enviado.

`false`

Si la máquina NO ha llegado a la posición a la cual se le ha enviado.

Parámetros

Este método no requiere parámetros de entrada.

14

```
bool IsBrakeOn()
```

Descripción general

Informa si está activo el freno.

Retorno

`true`
Si el variador ha aplicado el freno.
`false`
Si el variador NO ha aplicado el freno.

Parámetros

Este método no requiere parámetros de entrada.

15

```
bool IsControlInhibit()
```

Descripción general

Indica si el variador tiene activo el bit `ControlInhibit`.

Retorno

`true`
El variador está inhibido.
`false`
El variador NO está inhibido.

Parámetros

Este método no requiere parámetros de entrada.

16

```
bool IsWatchdogError()
```

Descripción general

Evalúa el bit de vida del variador para poder realizar una parada en caso de que este se quede bloqueado.

Retorno

`true`
El variador no cumple el protocolo de activación de su bit de vida, y por lo tanto puede ser que este funcionando incorrectamente.
`false`
El variador está operando correctamente con su bit de vida.

Parámetros

Este método no requiere parámetros de entrada.

17

```
bool IsWeighAvailable()
```

Descripción general

Indica cuando la maniobra de pesaje ha terminado y hay un peso disponible

Retorno

`true`

Existe un valor del peso disponible.
`false`
NO existe un valor del peso disponible.

Parámetros

Este método no requiere parámetros de entrada.

18

```
void Positioning(dword Position)
```

Descripción general

Da la orden de posicionamiento en manual.

Retorno

`void`
Esta función no retorna nada.

Parámetros

1. `dword` `Position`

   Valor de la posición a la que se desea enviar la máquina.

19

```
void PresetResolver(dword Preset)
```

Descripción general

Este método presetea el resolver (sólo tiene utilidad si el bucle de posicionamiento se cierra con el resolver). Sirve como preset ya que el valor del resolver puede perderse en muchas situaciones (pérdidas de tensión, caídas de bus, deslizamiento de ruedas).

Retorno

`void`
Este método no retorna nada

Parámetros

1. `dword` `Preset`

   Valor que se establecerá como actual al resolver.

20

```
void Reset()
```

Descripción general

Fuerza el rearme del variador.

Retorno

`void`
Este método no retorna nada

Parámetros

Este método no requiere parámetros de entrada.

21

```
void ResetCanBus()
```

Descripción general

Fuerza el rearranque del bus CAN del variador (telémetro) por si se hubiera quedado en un estado preoperacional.

Retorno

`void`

Este método no retorna nada

Parámetros

Este método no requiere parámetros de entrada.

22

```
void ResetWatchdog()
```

Descripción general

Resetea el error de watchdog.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

23

```
bool SafetyChainError()
```

Descripción general

Evalúa si existe un problema en el módulo de seguridades.

Retorno

`true`

Existe un problema en el módulo de seguridades. Una de las seguridades de la cadena se ha accionado.

`false`

La cadena de seguridades está correcta.

Parámetros

Este método no requiere parámetros de entrada.

24

```
byte SelectedEngine()
```

Descripción general

Retorna el número de motor seleccionado.

Retorno

`byte`

Retorna el motor que está seleccionado en ese momento:

0 → Ningún motor seleccionado

1 → Motor del eje 1

2 → Motor del eje 2

3 → Motor del eje 3

Parámetros

Este método no requiere parámetros de entrada.

25

```
void SelectEngine(byte Engine)
```

Descripción general

Selecciona el motor a controlar por el variador.

Retorno

```
void
```

Este método no retorna nada

Parámetros

```
1. byte Engine
```

Motor que se desea seleccionar con la siguiente codificación:

0 → Ningún motor seleccionado

1 → Motor del eje 1

2 → Motor del eje 2

3 → Motor del eje 3

26

```
void SetAutoAccelAndSpeed(byte Speed,
                          byte Accel)
```

Descripción general

Establece la velocidad y aceleración del motor en automatico.
Nota: a diferencia del modo manual, en este caso, el valor de velocidad es un porcentaje de la velocidad máxima posible.

Retorno

```
void
```

Esta función no retorna nada.

Parámetros

```
1. byte Speed
```

Valor de la velocidad. Se toma como valor de velocidad el valor absoluto del

parámetro pasado (es un porcentaje sobre el valor configurado en los parámetros internos del variador)

2. byte Accel

Valor de la aceleración. Al igual que la velocidad es un valor sobre el configurado en los parámetros internos del variador.

27

```
void SetManualAccel(byte Speed,
                    byte Accel)
```

Descripción general

Establece la velocidad y aceleración del motor en manual.

Retorno

void

Esta función no retorna nada.

Parámetros

1. byte Speed

Valor de la velocidad. Se toma como valor de velocidad el valor absoluto del parámetro pasado (es un porcentaje sobre el valor configurado en los parámetros internos del variador)

2. byte Accel

Valor de la aceleración. Al igual que la velocidad es un valor sobre el configurado en los parámetros internos del variador.

28

```
void SetSpeed(dword Speed)
```

Descripción general

Establece la velocidad y sentido del motor en manual.

Retorno

void

Esta función no retorna nada.

Parámetros

1. dword Speed

Valor de la velocidad. Se toma como valor de velocidad el valor absoluto del parámetro pasado (es un porcentaje sobre el valor configurado en los parámetros internos del variador), y se utiliza el signo para indicar el sentido de giro:

- Positivo → Avance

- Negativo → Retroceso

29

```
void SetSpeedPercent(dword Speed,
                     bool Sent)
```

Descripción general

Establece la velocidad y sentido del motor en manual. Es similar a `SetSpeedPercent`, pero establece el sentido a partir de un parámetro indenpendiente.

Retorno

`void`

Esta función no retorna nada.

Parámetros

1. `dword` Speed

Valor de la velocidad. Se toma como valor de velocidad el valor absoluto del parámetro pasado (es un porcentaje sobre el valor configurado en los parámetros internos del variador)

2. `bool` Sent

Sentido de giro:

- `True` → Positivo → Avance
- `False` → Negativo → Retroceso

30

```
void SpeedStop()
```

Descripción general

Este método detiene la máquina mediante un QSP.

Retorno

`void`

Este método no retorna nada

Parámetros

Este método no tiene parámetros de entrada.

31

```
void Stop()
```

Descripción general

Este método realiza una parada controlada en el variador.

Retorno

`void`

Esta función no retorna nada.

Parámetros

Este método no requiere parámetros de entrada.

32

```
void StopPreset()
```

Descripción general

Interrumpe el comando de preset de resolver.

Retorno

`void`

Este método no retorna nada

Parámetros

Este método no requiere parámetros de entrada.

33

```
void StopWeight()
```

Descripción general

Finaliza la maniobra de pesaje.

Retorno

`void`

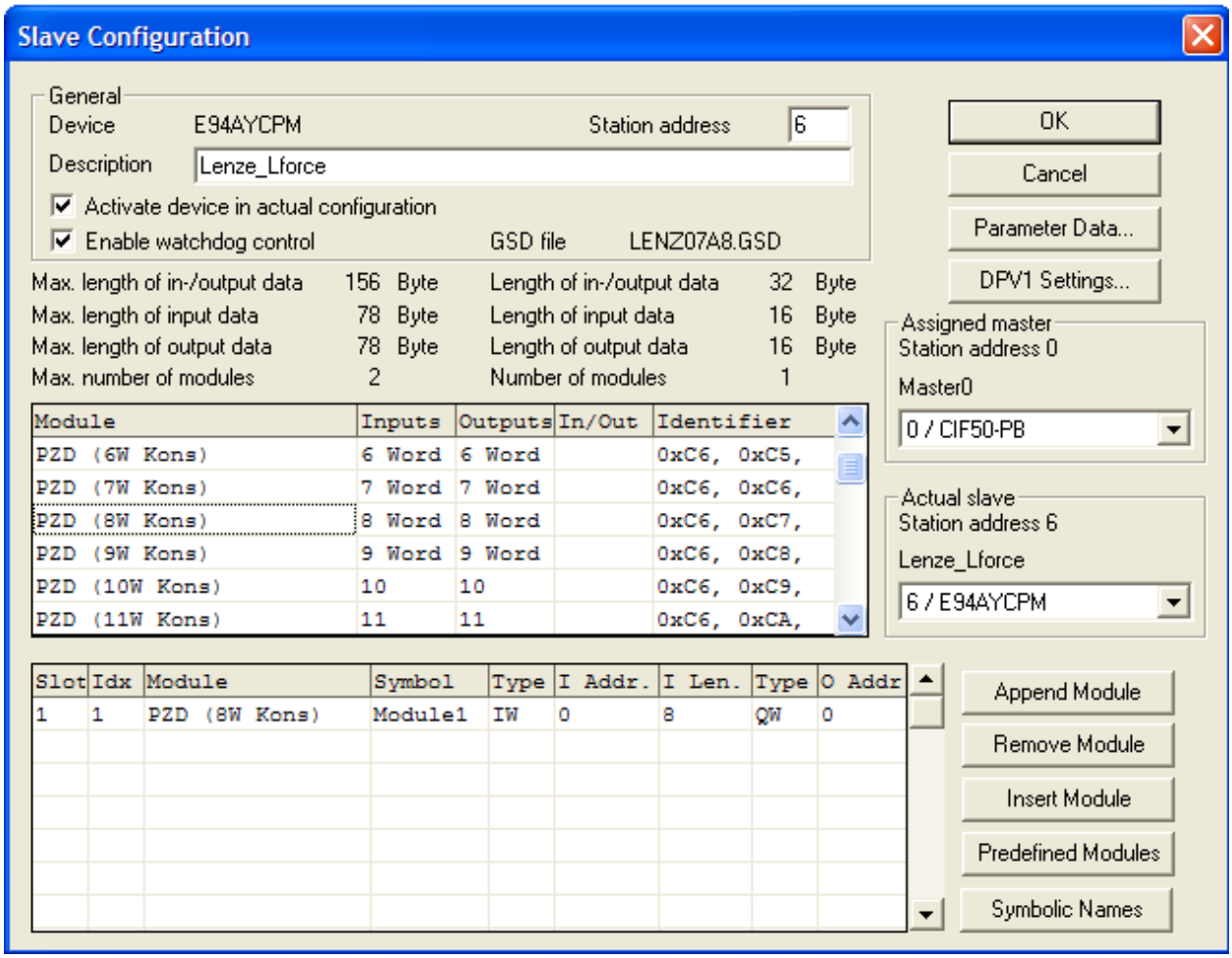
Este método no devuelve ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

4.2.13.2 Configuración periferia

La configuración que se ha de asignar a dicho variador para el correcto funcionamiento del componente es la indicada en la imagen:



4.2.14 LENZE: SMV VECTOR

4.2.14.1 Introducción

El componente `SMVector` permite gestionar los parámetros de configuración de los variadores SMVector de Lenze.

El componente solo está diseñado para la gestión de los parámetros, y no para el control del variador.

4.2.14.2 Parámetros

Este variador identifica sus parámetros mediante un número entero. Este índice será el utilizado en los métodos del componente para leer o escribir valores en el variador. Para más información acerca de los distintos parámetros del variador, consultar su manual o el manual del interface Profibus que utiliza.

Para la gestión de los parámetros, el variador utiliza la propia imagen de proceso. En concreto, utiliza un bloque de 4 `words` (8 `bytes`) de salidas para el envío de las peticiones de lectura/escritura y un bloque de 4 `words` (8 `bytes`) de entradas para la recepción de los ACK y las lecturas. La posición de estos bloques no es fija y

pueda ser configurada por el usuario. El variador permite 3 configuraciones posibles:

- *Sin control de parámetros*. Estos bloques no se mapean y por tanto no pueden ser utilizados.
- *Mapeados al inicio de la imagen de proceso*. Los bloques se mapean en las direcciones mas bajas de las asignasdas al dispositivo.
- *Mapeadas al final de la imagen de proceso*. Los bloques se mapean en las direcciones mas altas de las asignadas al dispositivo.

Estas opciones se controlan desde el parámetro 431.
Para que sea compatible con este componente, el valor de este parámetro tiene que ser 2, es decir, mapear los bloques al final de la imagen de proceso.

Este variador permite además configurar la información que muestra en la imagen de proceso aparte de los bloques del control de parámetros.
Para la compatibilidad con este componente, el tamaño de la información que se mapea antes de los bloques tiene que ser de 2 words de E/S.

Es decir, finalmente tendríamos configurado el variador de la siguiente manera:

Entrada

Word 0	Word 1	Word 2	Word 3	Word 4	Word 5
Status 1	Status 2	Bloque de control de parámetros			

Salida

Word 0	Word 1	Word 2	Word 3	Word 4	Word 5
Control 1	Control 2	Bloque de control de parámetros			

4.2.14.3 Interface

Este componente expone el siguiente interface

Class	SMVector
BUS IN	12 bytes
BUS OUT	12 bytes

Listado de métodos:

1	`bool ReadParameterRequest(word ParamIndex)`
2	`bool RequestSuccess(byte &ErrorCode)`
3	`bool WriteParameterRequest(word ParamIndex, word ParamValue)`

Descripción de los métodos:

1

```
bool ReadParameterRequest(word ParamIndex)
```

Descripción general

Envia al variador una petición de lectura del valor de un parámetro de su configuración interna.

Retorno

`true`
Es posible realizar la petición solicitada.
`false`
No se puede realizar la petición (por ejemplo, se encuentra realizando otra anterior).

Parámetros

1. `word` ParamIndex
   Índice del parámetro que se desea leer.

2

```
bool RequestSuccess(byte &ErrorCode)
```

Descripción general

Comprueba si la petición que se esta ejecutando a terminado con éxito.

Retorno

`true`
La operación solicitada (lectura o escritura) se ha realizado con éxito.
`false`
No se ha podido finalizar correctamente la lectura o escritura que se había solicitado.

Parámetros

1. `byte` ErrorCode
   Se indica el código de error en caso de haber terminado de forma incorrecta la operación se ha solicitado (lectura o escritura).

3

```
bool WriteParameterRequest(word ParamIndex,
                           word ParamValue)
```

Descripción general

Envia al variador una petición de escritura del valor de un parámetro de su configuración interna.

Retorno

`true`
Es posible realizar la petición solicitada.
`false`
No se puede realizar la petición (por ejemplo, se encuentra realizando otra anterior).

Parámetros

1. `word` ParamIndex
   Índice del parámetro que se desea escribir.
2. `word` ParamValue

Valor a asignar al parámetro que se desea escribir.

4.2.15 LENZE: VARIADOR 8400

4.2.15.1 Interface

Se ha desarrollado un componente que permite la comunicación con el variador Lenze 8400 para CANOPEN, que se está aplicando por ejemplo en la máquina Clasimat de MECALUX S.A.

El interface del componente es el siguiente:

Class	Lenze8400
BUS IN	16 bytes
BUS OUT	16 bytes

Listado de métodos:

1	dword CurrentAcell (dword Numerator, dword Denominator)
2	dword CurrentPosition()
3	dword CurrentSpeed(dword Numerator, dword Denominator)
4	dword CurrentTorque()
5	bool Error()
6	word GetErrorCode(word &Module)
7	bool GetInputSignal(byte Index)
8	void GoToHome()
9	void GoToPosition(dword Position)
10	bool HomePositionAvalaible()
11	void Inhibit(bool Inhibit)
12	bool InPosition()
1	bool IsBrakeOn()

3	
14	bool IsControlInhibit()
15	void IsPresetAvalaible()
16	bool IsWatchdogError()
17	void Positioning(dword Position)
18	void PresetResolver(dword Preset)
19	void Ready()
20	void Reset()
21	void ResetWatchdog()
22	bool SafetyChainError()
23	byte SelectedEngine()
24	void SelectEngine(byte Engine)
25	void SetAutoSpeed(byte Speed)
26	void SetManualSpeed(byte Speed)
27	void SetOutputSignal(byte Index, bool Value)
28	void SetSpeed(dword Speed)
29	void SetSpeedPercent(dword Speed, bool Sent)
3	void SpeedStop()

0	
31	void Stop()

Descripción de los métodos:

```
dword CurrentAcell(dword Numerator,
                   dword Denominator)
```

Descripción general

Informa de la aceleración actual a la que se mueve la máquina.

Retorno

dword

Retorna la velocidad actual a la que se mueve la máquina, multiplicada por los factores de conversión, de la siguiente forma:

Aceleración * Numerator/Denominator

Parámetros

1. dword Numerator

   Factor de conversión numerador.

2. dword Denominator

   Factor de conversión denominador.

```
dword CurrentPosition()
```

Descripción general

Devuelve la posición actual de la máquina dada por el variador.

Retorno

dword

Retorna el valor de la posición actual.

Parámetros

Este método no requiere parámetros de entrada.

```
dword CurrentSpeed(dword Numerator,
                   dword Denominator)
```

Descripción general

Informa de la velocidad actual a la que se mueve la máquina.

Los parámetros Numerador y Denominador son un factor de conversión para mostrar el valor en el formato deseado.

La velocidad que devuelve, el componente la calcula siempre como unidades de posición / mseg.,

Si por ejemplo el variador entrega la posición en mm. el variador devolverá una posición en mm/mseg. multiplicado por el factor de conversión que resulta de Numerador/Denominador.

Por ejemplo, si el cálculo de la velocidad resulta ser de 3, y los parámetros enviados son Numerador = 1 y Denominador = 1, la velocidad entregada será de 3 mm/mseg, es decir, de 3 m/seg.

Si el cálculo de la velocidad es 3,5 y Numerador = 1 y Denominador = 1 el resultado será que el componente devuelve 3, sin embargo si Numerador = 10 y Denominador = 1 el resultado es de 35.

Retorno

dword

Retorna la velocidad actual a la que se mueve la máquina, multiplicada por los factores de conversión, de la siguiente forma:

Velocidad * `Numerator/Denominator`

Parámetros

1. `dword Numerator`

   Factor de conversión numerador.

2. `dword Denominator`

   Factor de conversión denominador.

`dword` **`CurrentTorque`**`()`

Descripción general

Este método devuelve el par del variador.

Retorno

`dword`

Retorna el valor del par.

Parámetros

Este método no requiere parámetros de entrada.

`bool` **`Error`**`()`

Descripción general

Este método indica si el variador está o no en error. Sirve para identificar los errores que se producen en el variador. Estos errores se rearman mediante el método `Reset`.

Retorno

`true`

Si el variador se encuentra en error.

`false`

Si el variador NO se encuentra en error.

Parámetros

Este método no requiere parámetros de entrada.

`word` **`GetErrorCode`**`()`

Descripción general

Devuelve el código de error del variador.

Retorno

`word`

Retorna el valor del código de error del variador. Dicho valor podrá ser consultado en la documentación específica de Lenze.

Parámetros

Este método no requiere parámetros de entrada.

```
bool GetInputSignal(byte Index)
```

Descripción general

Informa del estado de sensores que están cableados directamente al variador y de los cuales el sistema de control necesita tener conocimiento.

Retorno

`bool`

Retorna el estado del sensor correspondiente que ha sido indicado mediante el parámetro `Index`. La lógica será negada:

Value	Descripción
0	*Sensor actuado*
1	*Sensor libre*

Parámetros

1. `byte Index`

   Índice de la señal de la cual se desea conocer su estado. En el Clasimat, la

   correspondencia sería la siguiente:

Señal	Sensor	Index
InputSignal1	*Final carrera Homing*	*1*
Reserved	*---*	*2*

```
void GoToHome()
```

Descripción general

Indica al variador la orden de realizar un Homing. Es utilzado para el caso de que el variador pierda su referencia de resolver.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

```
void GoToPosition(dword Position)
```

Descripción general

Da la orden de posicionamiento en automático.

Retorno

`void`

Esta función no retorna nada.

Parámetros

2. `dword Position`

   Valor de la posición a la que se desea enviar la máquina.

```
bool HomePositionAvalaible()
```

Descripción general

Indica asi el variador ya ha finalizado la maniobra de Homing y si ha realizado el preset al resolver interno correctamente.

Retorno

`true`

La maniobra y el preset han finalizado con éxito.

`false`

La maniobra y el preset aún no han terminado.

Parámetros

Este método no requiere parámetros de entrada.

```
void Inhibit(bool Inhibit)
```

Descripción general

Este método realiza la inhibición del variador directamente mediante CW.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `bool Inhibit`

   Bit de inhibición del variador.

```
bool InPosition()
```

Descripción general

Esta función retorna si la máquina ha llegado o no a la posición de destino. Esta función dejará de marcar `InPosition` aunque lo este realmente en el momento que se invoque el método `Stop`.

Retorno

`true`

Si la máquina ha llegado a la posición a la cual se le ha enviado.

`false`

Si la máquina NO ha llegado a la posición a la cual se le ha enviado.

Parámetros

Este método no requiere parámetros de entrada.

```
bool IsBrakeOn()
```

Descripción general

Informa si está activo el freno.

Retorno

`true`

Si el variador ha aplicado el freno.

`false`

Si el variador NO ha aplicado el freno.

Parámetros

Este método no requiere parámetros de entrada.

`bool` **`IsControlInhibit`**`()`

Descripción general

Indica si el variador tiene activo el bit `ControlInhibit`.

Retorno

`true`

El variador está inhibido.

`false`

El variador NO está inhibido.

Parámetros

Este método no requiere parámetros de entrada.

`bool` **`IsPresetAvalaible`**`()`

Descripción general

Este método retorna si se ha finalizado o no de hacer el preset.

Retorno

`true`

Se ha realizado correctamente el preset

`false`

Aun no se ha completado el preset

Parámetros

Este método no requiere parámetros de entrada.

```
bool IsWatchdogError()
```

Descripción general

Evalúa el bit de vida del variador para poder realizar una parada en caso de que este se quede bloqueado.

Retorno

`true`

El variador no cumple el protocolo de activación de su bit de vida, y por lo tanto puede ser que este funcionando incorrectamente.

`false`

El variador está operando correctamente con su bit de vida.

Parámetros

Este método no requiere parámetros de entrada.

```
void Positioning(dword Position)
```

Descripción general

Da la orden de posicionamiento en manual.

Retorno

`void`

Esta función no retorna nada.

Parámetros

2. dword Position

Valor de la posición a la que se desea enviar la máquina.

```
{
```

```
void PresetResolver(dword Preset)
```

Descripción general

Este método solicita un preset al valor indicado en el parámetro `Preset`

Retorno

`void`

Este método no retorna nada

Parámetros

1. dword Preset

Valor que se quiere dar para hacer el preset.

```
{
```

```
void Ready()
```

Descripción general

Este método informa del estado "Ready" del variador.

Retorno

`void`

Este método no retorna nada

Parámetros

Este método no requiere parámetros de entrada.

```
{
```

```
void Reset()
```

Descripción general

Este método realiza el reset del variador. Es utilizado para realizar el rearme del variador en caso de que entre en error.

Retorno

`void`

Este método no retorna nada

Parámetros

Este método no requiere parámetros de entrada.

```
}
```

```
void ResetWatchdog()
```

Descripción general

Resetea el error de watchdog.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

```
}
```

```
bool SafetyChainError()
```

Descripción general

Evalúa si existe un problema en el módulo de seguridades.

Retorno

`true`

Existe un problema en el módulo de seguridades. Una de las seguridades de la cadena se ha accionado.

`false`

La cadena de seguridades está correcta.

Parámetros

Este método no requiere parámetros de entrada.

```
}
```

```
byte SelectedEngine()
```

Descripción general

Retorna el número de motor seleccionado.

Retorno

byte

Retorna el motor que está seleccionado en ese momento, según la siguiente codificación:

Engine selected()	Engine Selected()	Motor seleccionado **CLASIMAT**
0	*0*	*Elevación - Eje Y*
1	*0*	*Extractor - Eje Z*
0	*1*	*Puerta*
1	*1*	*Sin selección - Libre*

Parámetros

Este método no requiere parámetros de entrada.

```
void SelectEngine(byte Engine)
```

Descripción general

Selecciona el motor a controlar por el variador.

Retorno

void

Este método no retorna nada.

Parámetros

1. `byte` `Engine`

Motor que se desea seleccionar con la siguiente codificación:

Engine selected()	Engine Selected()	Motor seleccionado **CLASIMAT**
0	*0*	*Elevación - Eje Y*
1	*0*	*Extractor - Eje Z*
0	*1*	*Puerta*
1	*1*	*Sin selección - Libre*

```
void SetAutoSpeed(byte Speed)
```

Descripción general

Establece la velocidad del motor en automático. Iría asociado junto con el método `GoToPosition()` para establecer la velocidad a la que se desea mover.

Retorno

`void`

Esta función no retorna nada.

Parámetros

1. `byte` `Speed`

Valor de la velocidad.

La velocidad será indicada en %, que se aplicará sobre un % configurado de la

velocidad máxima en VF (considerando siempre 100% → 0x4000h)

```
void SetManualSpeed(byte Speed)
```

Descripción general

Esta función establece la velocidad del motor en modo manual. Iría asociada junto con el método `Positioning` para establecer la velocidad a la que se desea mover.

Retorno

`void`

Esta función no retorna nada.

Parámetros

1. `byte Speed`

   Valor de la velocidad.

   La velocidad será indicada en %, que se aplicará sobre un % configurado de la

   velocidad máxima en VF (considerando siempre 100% → 0x4000h)

`void SetOutputSignal(byte Index, bool Value)`

Descripción general

Este método informa al variador del estado de señales que han sido llevadas al Sistema de Control y no al variador.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `byte Index`

Índice de la señal que se desea informar. En el Clasimat, la correspondencia

es la siguiente:

Señal	Sensor	Index
OutputSignal1	*Final de carrera superior*	*1*
OutputSignal2	*Final de carrera inferior*	*2*

2. `bool Value`

Indica el estado del sensor referenciado en `Index`, siempre con lógica negada:

Value	Descripción
0	*Sensor actuado*
1	*Sensor libre*

```
void SetSpeed(dword Speed)
```

Descripción general

Establece la velocidad y sentido del motor en manual.

Retorno

`void`

Esta función no retorna nada.

Parámetros

1. dword Speed

Valor de la velocidad. Se toma como valor de velocidad el valor absoluto del

parámetro pasado (es un porcentaje sobre el valor configurado en los

parámetros internos del variador), y se utiliza el signo para indicar el sentido

de giro:

a. Positivo → Avance
b. Negativo → Retroceso

```
void SetSpeedPercent(dword Speed,
                     bool Sent)
```

Descripción general

Establece la velocidad y sentido del motor en manual. Es similar a `SetSpeedPercent()`, pero establece el sentido a partir de un parámetro indenpendiente.

Retorno

`void`

Esta función no retorna nada.

Parámetros

1. dword Speed

   Valor de la velocidad. Se toma como valor de velocidad el valor absoluto del

   parámetro pasado (es un porcentaje sobre el valor configurado en los parámetros internos del variador)

2. bool Sent

   Sentido de giro:

   a. True → Positivo → Avance
   b. False → Negativo → Retroceso

```
void SpeedStop()
```

Descripción general

Este método detiene la máquina mediante un QSP.

Retorno

```
void
```

Este método no retorna nada

Parámetros

Este método no tiene parámetros de entrada.

```
void Stop()
```

Descripción general

Este método realiza una parada controlada en el variador.

Retorno

void
Esta función no retorna nada.
Parámetros
Este método no requiere parámetros de entrada.

4.2.15.2 Configuración periferia

La configuración que se ha de asignar a dicho variador para el correcto funcionamiento del componente es con el EDS suministrado por Lenze que permite utilizar 2 PDOs, tanto de entrada como de salida.

4.2.16 ESCÁNER: Interface ISCANNER

4.2.16.1 Concepto de Interface universal

Después de analizar el diferente hardware asociado a una instalación se llega a la conclusión que desde el punto de vista de control durante la duración de la vida de una instalación es posible que el hardware cambie al agotarse los repuestos, dejar de fabricarse determinado material, etc.

Sin embargo desde el punto de vista de la aplicación de Control, estos cambios implican más cosas, ya que los interfaces para manejar dichos dispositivos son diferentes, normalmente no muy diferentes pero si sutilmente diferentes.

De forma similar cuando se desarrolla software de control genérico (por ejemplo un transportador de PIE), que tiene hardware asociado, debe ser aislado de modelo de hardware que se instale. Es decir si el escáner es Sick conectado al puerto serie o Sick conectado a TCP o Data Control no debería influir en como se programa el secuenciador.

Para evitar este caso se irán creando interfaces genéricos que no pueden ser utilizados nada más que para aislar tipos de componentes entre sí.

Estos interfaces se utilizan para definir parámetros del secuenciador, pero nunca se utilizarán para declarar una variable en una máquina de este tipo, ya que no se asocian a hardware ninguno, solamente definen una forma de trabajar. Posteriormente al declarar una máquina que utilice este secuenciador le asociaremos alguno de los componentes compatibles con dicho interface.

4.2.16.2 Interface

Los datos de creación del componente son los siguientes:

Class	IScanner
BUS IN	0 bytes

BUS OUT 0 bytes

Listado de métodos:

1	`bool ConfigScanner(byte Parameter1, string Parameter2)`
2	`byte GetData(byte Index)`
3	`dword GetStatus()`
4	`void Reset()`
5	`void StartRead()`

Descripción de los métodos:

1

```
bool ConfigScanner(byte Parameter1,
                   string Parameter2)
```

Descripción general

Este método debe invocarse en las rutinas de arranque, permite abrir el puerto serie de comunicaciones para empezar a utilizar el escáner.

Retorno

`true`
La operación se ha realizado con éxito.
`false`
La operación NO se ha realizado con éxito.

Parámetros

1. `byte Parameter1`
   El significado de este parámetro depende de la implementación, es necesario consultar el componente asociado.
2. `string Parameter2`
   El significado de este parámetro depende de la implementación, es necesario consultar el componente asociado.

2

```
byte GetData(byte Index)
```

Descripción general

Cuando la lectura del escáner es correcta los datos leídos se almacenan en un buffer interno. Este método permite recuperar estos datos del mismo.

Retorno

`byte`
Retorna el valor del contenido del buffer en la posición indicada dentro del parámetro de entrada `Index`.

Parámetros

1. `byte Index`
   Índice del dato que se quiere recuperar del buffer.

3

```
dword GetStatus()
```

Descripción general

Este método sirve para obtener el estatus de lectura del escáner.

Retorno

dword

Retorna un valo que indica el estado en el que se encuentra el lector:

a. 1 → Código de barras leído correctamente.
b. 2 → Error de lectura de la etiqueta.
c. 3 → "Basura" en el puerto de comunicaciones.
d. 4 → Desbordamiento del buffer.

Parámetros

Este método no requiere parámetros de entrada.

4

```
void Reset()
```

Descripción general

Este método limpia el buffer de lectura de sus datos actuales. Es recomendable realizar un `Reset` entre lecturas.

Retorno

void

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

5

```
void StartRead()
```

Descripción general

Este método inicia la lectura en los escáners en los que sea necesario dispararla desde el programa de control y no mediante señal eléctrica.

Retorno

void

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

4.2.17 SICK: ESCÁNER PUERTO SERIE

4.2.17.1 Introducción

Para poder realizar lecturas de etiquetas se ha desarrollado un componente que realiza el interface con la familia de escáneres de Sick CLV. El Interface

desarrollado funciona a través del cable serie. En las siguientes secciones explicaremos tanto la configuración del scanner como lo métodos expuestos por el componente.
Este componente soporta el Interface IScanner.

4.2.17.2 Configuración del escáner

4.2.17.2.1 Alcance

Este apartado debería servir como referencia a la hora de instalar y configurar el escáner CLV220 de SICK a fin de usarlo en una instalación con el sistema Galileo implantado.

Sin embargo, no debe tomarse como un manual de referencia para la aplicación CLV Setup, puesto que sólo se limita a indicar los parámetros concretos que se han de modificar mediante dicha aplicación, sin entrar en detalle en el funcionamiento de la misma. A tal fin, dicha aplicación viene con una documentación en paginas HTML que se puede instalar a la vez que la aplicación (recomendado).

4.2.17.2.2 Elementos Hardware

El escáner CLV220 viene acompañado de una caja de conexiones que habremos de cablear de forma adecuada a fin de realizar tanto la programación del dispositivo como de conectarlo a un host (habitualmente un PC con una aplicación que espera la lectura de códigos de barras).

Todo este cableado se realiza mediante puertos serie, el paquete suele incluir un cable serie para conectar el escáner con la caja de conexiones, y necesitamos otro cable serie con dos conectores hembra para enlazar la caja de conexiones con el puerto serie del PC desde el que queremos programar el dispositivo. Este segundo cable no va incluido en el paquete, así que tendremos que obtenerlo aparte. Dicho cable sólo será necesario mientras queramos programar el escáner, no es necesario que esté conectado para el funcionamiento normal del dispositivo.

4.2.17.2.3 Comunicación vía serie

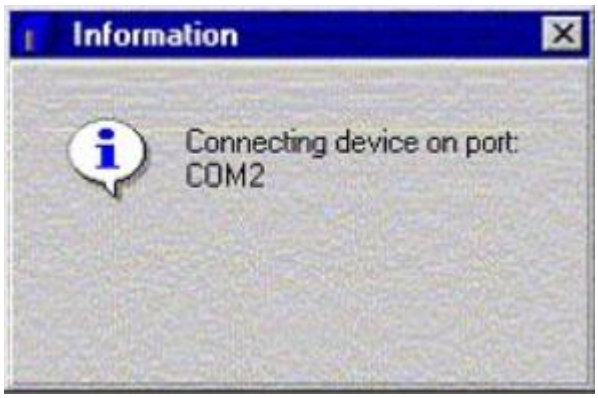
Para la conexión del escáner con el dispositivo host necesitaremos además un tercer cable serie, con un conector hembra en uno de los extremos, que conectaremos al PC. En el otro extremo deberemos conectar los hilos 2, 3 y 5 a los terminales adecuados de la caja de conexiones según la tabla que se muestra a continuación:

Pin	*Descripción*	*Conectar a*
5	Gnd (Señal Gnd para RS 232)	Gnd, PC pin 5 (9 pin plug)
10	RxD Host (RS 232)	TxD, PC pin 3 (9 pin plug)
11	TxD Host (RS 232)	RxD, PC pin 2 (9 pin plug)

En la parte izquierda de la tabla aparecen los pins de la caja de conexiones que deben enlazar con los hilos del cable serie que se muestran en la parte derecha de la tabla. Una vez realizadas las conexiones, podemos empezar a configurar el funcionamiento del escáner.

4.2.17.2.4 Inicio del CLV Setup

Al arrancar el programa, una pequeña pantalla nos indica que la aplicación está intentando conectarse y extraer la información de configuración del dispositivo:

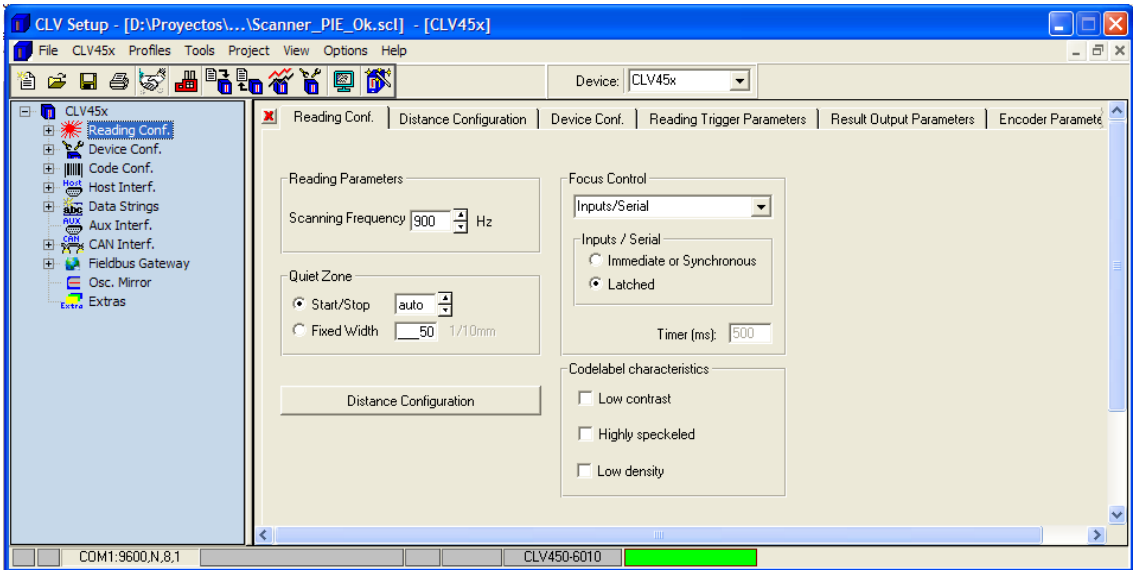


A continuación aparecerá la pantalla principal de la aplicación. El resultado del intento de comunicación se muestra en la parte inferior derecha, mediante una barra indicadora del estado de la conexión. Si dicho estado es "conectado", el programa obtendrá la configuración actual del dispositivo y la mostrará en pantalla. En otro caso, se dará un mensaje de error y se desactivarán la mayoría de las opciones de la aplicación.

4.2.17.2.5 Pantalla principal, configuración de la lectura

La aplicación muestra la información de configuración separada en varias pestañas según la funcionalidad en que se encuadren.

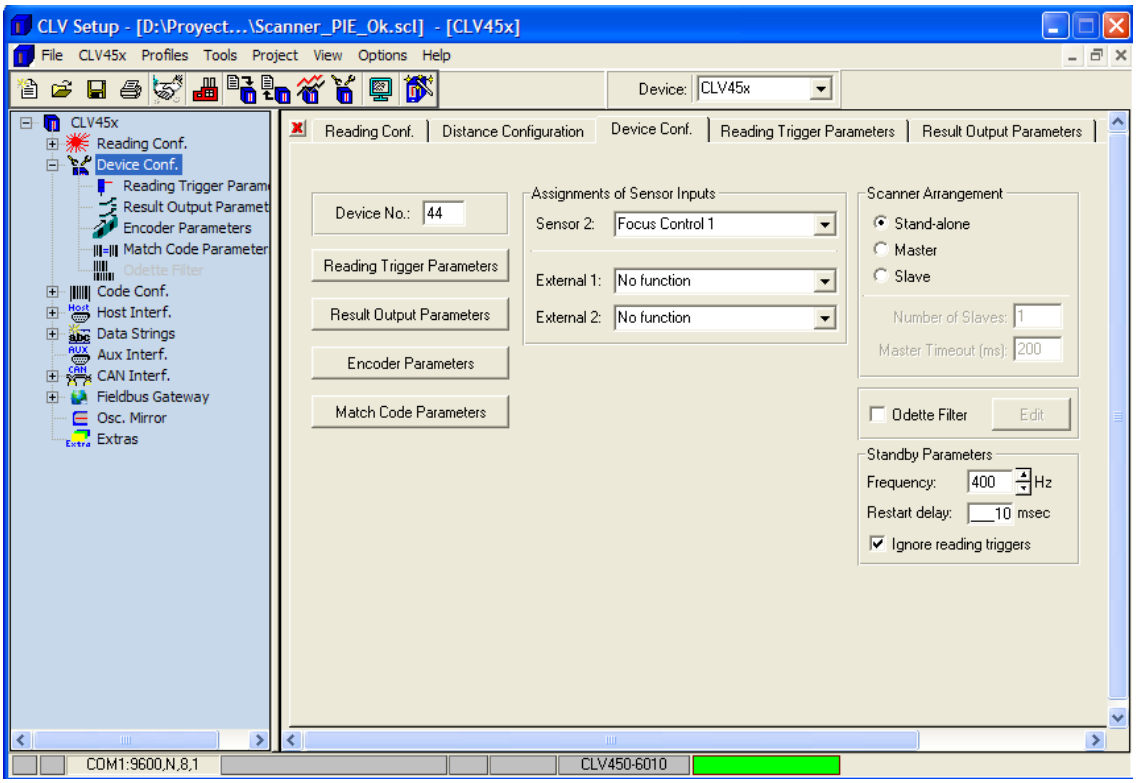
La pestaña de configuración de lectura es la pestaña visible por defecto en cuanto arranca la aplicación. Permite modificar los parámetros del escáner que afectan cómo este dispositivo lee los códigos de barras. Los valores por defecto que la aplicación lee del escáner deberían ser suficientemente apropiados, como se ve en la siguiente imagen:



Es importante resaltar que si la detección automática no lo ha hecho, deberemos establecer nosotros mismos en el combo de la parte superior derecha el modelo de escáner que estamos configurando (CLV220).

4.2.17.2.6 Configuración del dispositivo

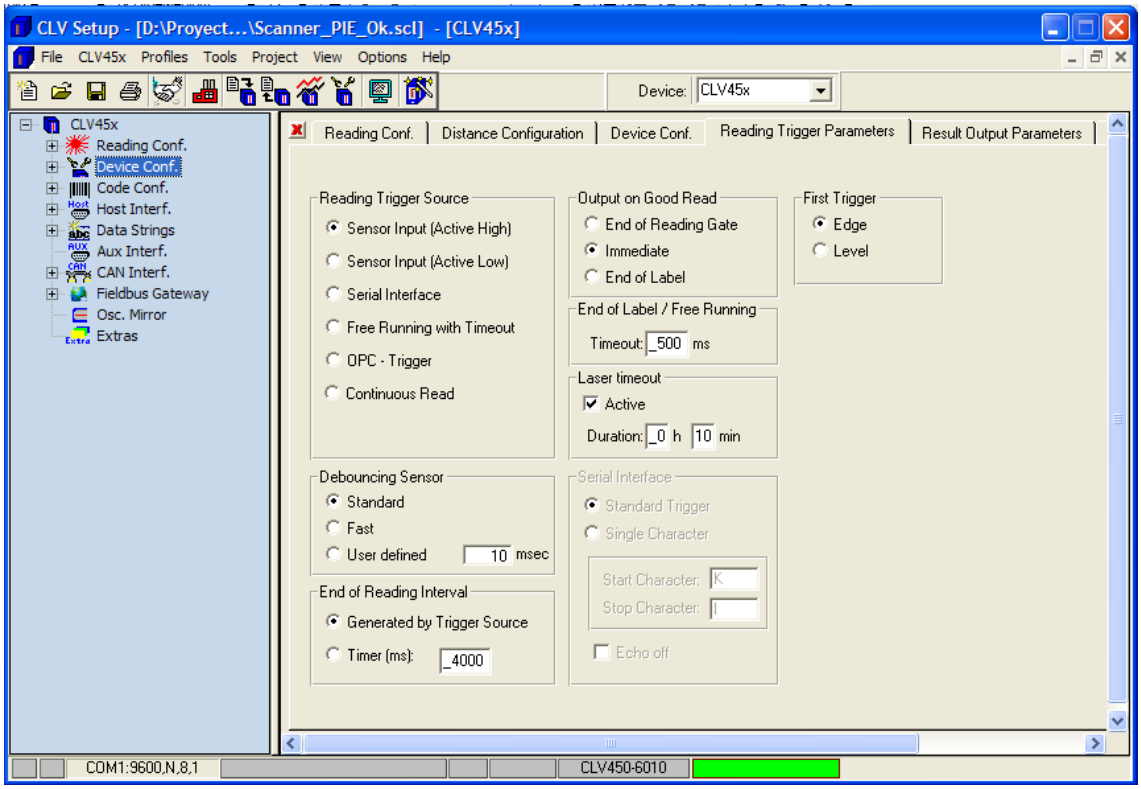
La segunda pestaña por la izquierda permite modificar los parámetros de lectura del escáner.



Aquí basta con asegurarnos que el escáner use el modo de funcionamiento apropiado, distinguiéndose dos casos:

- *Modo de lectura desatendido:* En este caso, hay que seleccionar el modo "*Free Running with Timeout*" pulsando el botón "*Edit Reading Trigger*" y en la ventana que aparecerá seleccionar el radio-botón que indica dicho modo. Este modo es el usado cuando queremos que el haz de lectura del escáner esté activado de continuo.
- *Modo de lectura por sensor:* Éste es el modo que debemos usar cuando disponemos de un sensor que detecta la presencia o llegada de una etiqueta, y queremos que el escáner active el haz de lectura cuando dicho sensor nos de una señal. En dicho caso, la opción a seleccionar de la ventana "*Edit reading Trigger*" es la marcada como "Sensor Input (Active High)" ó "*Sensor Input (Active Low)*" según el valor del sensor que indique la presencia. En ambos modos deberemos dejar la opción "Output *of Good Read*" en el valor "*Inmediate*", y el valor "Free Running" a 500 ms debería ser suficiente.

A continuación se muestra dicha pantalla con los valores correctos para el modo de lectura desatendido.



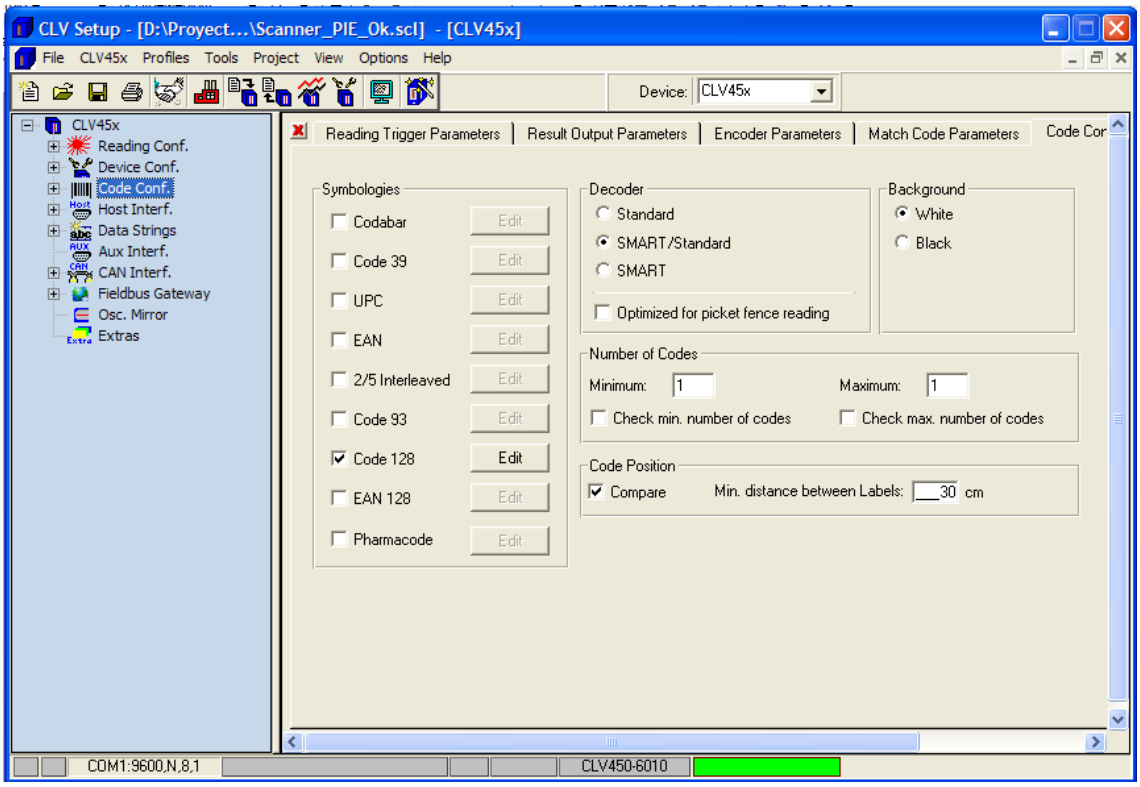
4.2.17.2.7 Configuración de los códigos a leer

La tercera pestaña por la izquierda permite activar o desactivar los tipos de códigos de barras que se van a leer desde el escáner, pudiendo, de esta manera, permitir o evitar que un tipo concreto de código de barras sea reconocido por el escáner.

El único código que NO debe ser activado es el "*Pharmacode*", el resto de los códigos pueden estar activados.

El decodificador debe estar en modo estándar.

El número de códigos máximo y mínimo debe ser en ambos casos '1'.

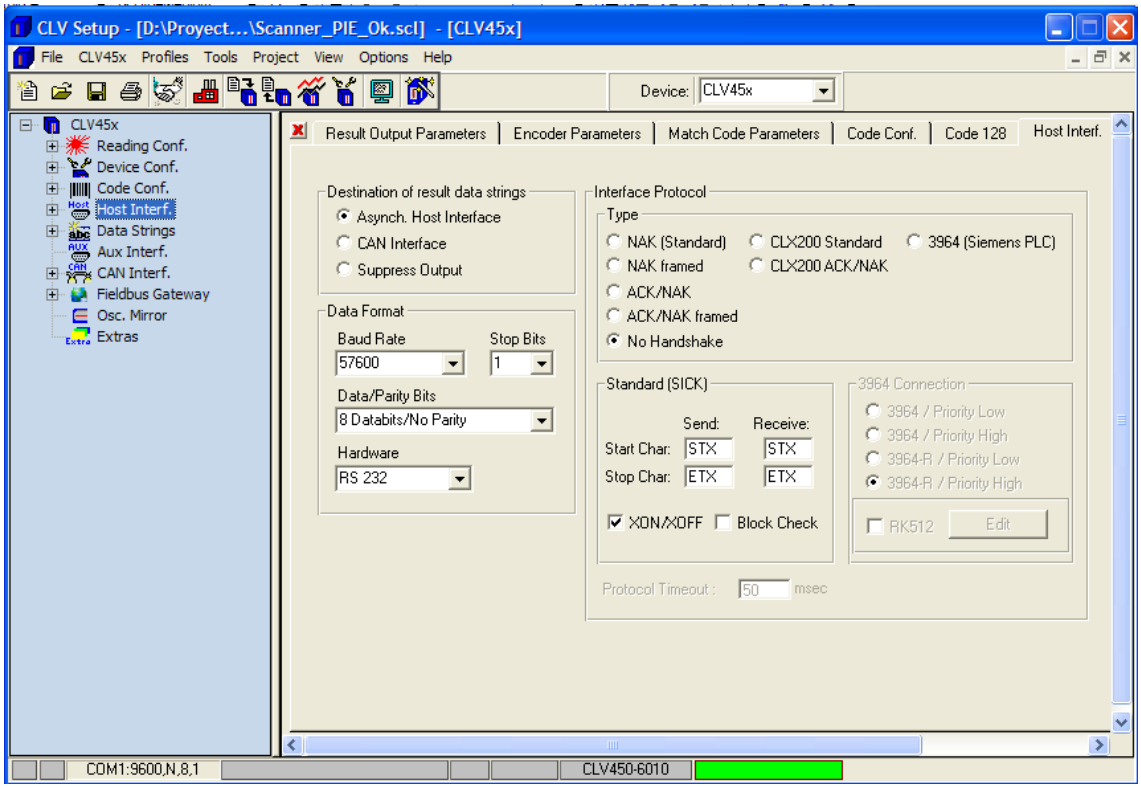


4.2.17.2.8 Configuración del Interface con el Host

El protocolo con el interfaz debe establecerse a No Handshake, y se deben usar los caracteres estándar de comienzo y final de mensaje (STX y ETX), además de habilitar el envío de XON/XOFF.

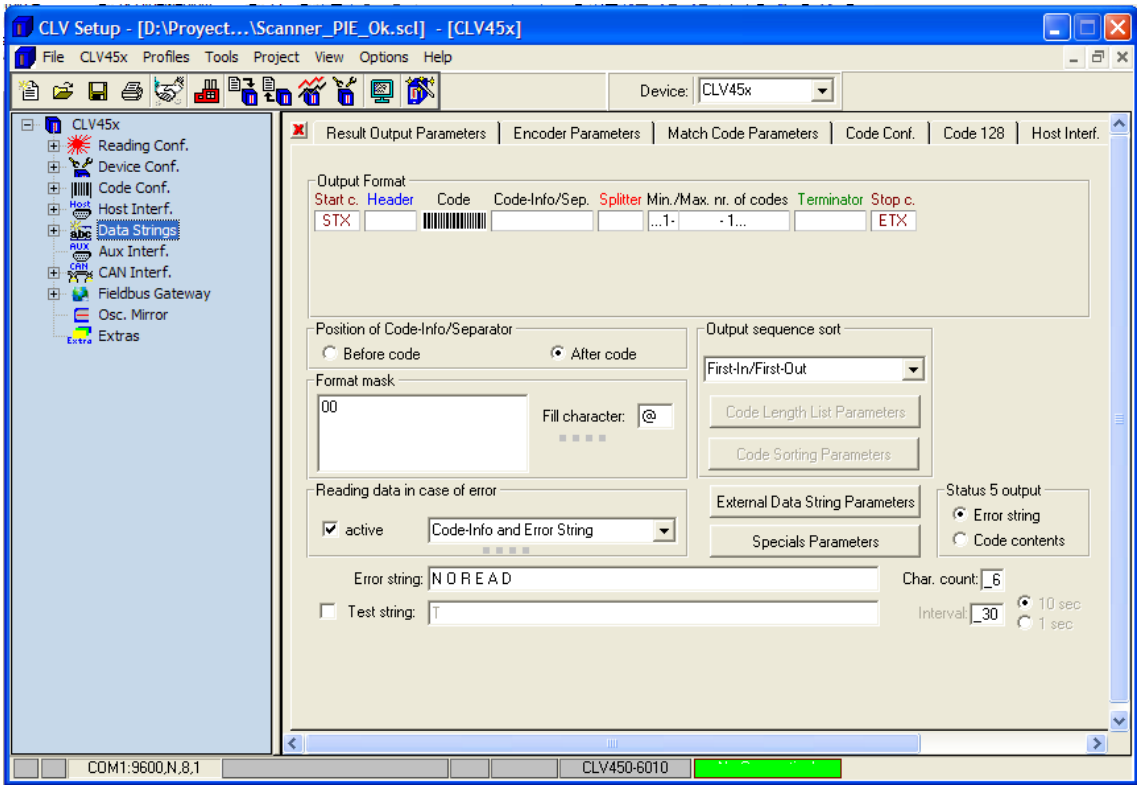
La configuración del puerto se establecerá a una velocidad de 38400 baudios, 1 bit de stop, 8 bits de datos sin paridad y con el Hardware RS-232.

A continuación se muestra dicha pantalla con los valores recomendados.



4.2.17.2.9 Configuración de las cadenas de datos

Esta pestaña permite configurar el mensaje que recibirá el Host cuando el escáner lea un código de barras de forma correcta o incorrecta. Es importante aquí que una serie de opciones estén definidas de forma adecuada, pues en otro caso se producirán errores al comunicarse con las aplicaciones del Host.



- No debe definirse cadena de cabecera, ni de separador ni de terminación.
- La máscara de formato debe ser vacía (Aparecerá un 00).
- El campo "*Output Sequence Sort*" debe de establecerse a "*First In/First Out*".
- El combo "*Wrong Read Format*" debe establecerse al valor "*Error String Only*"
- La cadena de error debe ser "*ERROR*" (en mayúsculas), sin caracteres de terminación extraños.

4.2.17.3 Interface

Los datos de creación del componente son los siguientes:

Class	`SickClv`
BUS IN	0 bytes
BUS OUT	0 bytes

El funcionamiento del mismo se basa en un thread en background que recoge constantes lecturas del scanner, si una lectura es buena se almacena en el buffer, si una lectura es mala se borra el buffer. El usuario debe consultar el status para decidir si los datos que se encuentran en el buffer corresponden a una lectura correcta.

Listado de métodos:

```
1  bool ConfigScanner(byte Port,
```

	string Parameter)
2	byte GetData(byte Index)
3	dword GetStatus()
4	void Reset()

Descripción de los métodos:

1

```
bool ConfigScanner(byte Port,
                   string Parameter)
```

Descripción general

Este método debe invocarse en las rutinas de arranque, permite abrir el puerto serie de comunicaciones para empezar a utilizar el escáner.

Retorno

`true`
La operación se ha realizado con éxito.
`false`
La operación NO se ha realizado con éxito.

Parámetros

1. `byte` Port
   Puerto de comunicaciones COM que se desea inicializar (normalmente de 1 a 4).
2. `string` Parameter
   Configuración de velocidad y datos de transmisión. Para la configuración de escáner recomendada en los puntos anteriores sería de la forma "`38400,n,8,1`"

2

```
byte GetData(byte Index)
```

Descripción general

Cuando la lectura del escáner es correcta los datos leídos se almacenan en un buffer interno. Este método permite recuperar estos datos del mismo.

Retorno

`byte`
Retorna el valor del contenido del buffer en la posición indicada dentro del parámetro de entrada `Index`.

Parámetros

2. `byte` Index
   Índice del dato que se quiere recuperar del buffer.

3

```
dword GetStatus()
```

Descripción general

Este método sirve para obtener el estatus de lectura del escáner.

Retorno

dword

Retorna un valo que indica el estado en el que se encuentra el lector:

a. 1 → Código de barras leído correctamente.
b. 2 → Error de lectura de la etiqueta.
c. 3 → "Basura" en el puerto de comunicaciones.
d. 4 → Desbordamiento del buffer.

Parámetros

Este método no requiere parámetros de entrada.

4

```
void Reset()
```

Descripción general

Este método limpia el buffer de lectura de sus datos actuales. Es recomendable realizar un `Reset` entre lecturas.

Retorno

void

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

4.2.18 SICK: ESCÁNER SICK TCP

4.2.18.1 Introducción

Para poder realizar lecturas de etiquetas se ha desarrollado un componente que realiza el interface con la familia de escáner de Sick CLV vía TCP. En las siguientes secciones explicaremos tanto la configuración del scanner como lo métodos expuestos por el componente.
Este componente soporta el Interface IScanner.

4.2.18.2 Configuración del escáner

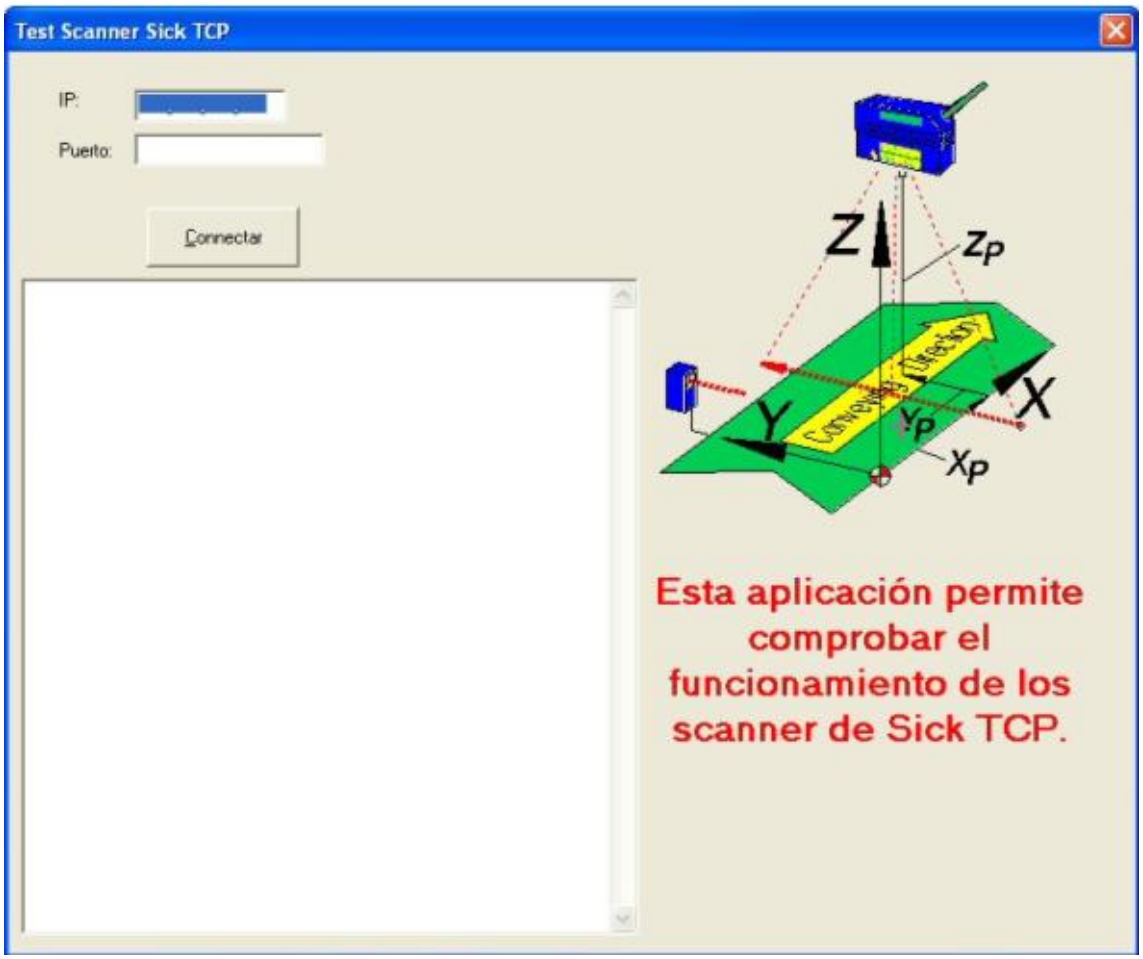
4.2.18.3 Test del escáner

Para determinar si el escáner será correctamente interpretado por Galileo, existe una utilidad de prueba, que es la que se adjunta con este documento. Esta utilidad no realiza el disparo del escáner, pero si realiza la comprobación de la lectura y el protocolo utilizado por Galileo.

En esta aplicación basta con seleccionar la dirección IP del scanner y el puerto de comunicaciones configurado. Una vez lanzada la lectura (normalmente por hardware forzando la señal de lectura), posteriormente a haber pulsado el botón de conectar y que nos indique que ha conectado con éxito, debiéramos ver aparecer

en pantalla las lecturas. Si esto no es así la configuración del scanner no es correcta, con lo cual no podríamos utilizarlo con Galileo.

En siguiente enlace te proporciona acceso a la herramienta de test:



4.2.18.4 Interface

Los datos de creación del componente son los siguientes:

Class	SickTCP
BUS IN	0 bytes
BUS OUT	0 bytes

El funcionamiento del mismo se basa en un thread en background que recoge constantes lecturas del scanner, si una lectura es buena se almacena en el buffer, si una lectura es mala se borra el buffer. El usuario debe consultar el status para decidir si los datos que se encuentran en el buffer corresponden a una lectura correcta.

Listado de métodos:

1	`bool ConfigScanner(byte Port, string Host)`
2	`byte GetData(byte Index)`
3	`dword GetDataLength()`
4	`dword GetStatus()`
5	`void Reset()`

Descripción de los métodos:

1

```
bool ConfigScanner(dword Port,
                   string IP)
```

Descripción general

Este método debe invocarse en las rutinas de arranque, y determina los datos de comunicación del escáner

Retorno

`true`

Este método siempre retorna `true`.

Parámetros

1. `dword Port`
   Puerto de comunicaciones.
2. `string IP`
   Dirección IP del escáner.

2

```
byte GetData(byte Index)
```

Descripción general

Cuando la lectura del escáner es correcta los datos leídos se almacenan en un buffer interno. Este método permite recuperar estos datos del mismo.

Retorno

`byte`

Retorna el valor del contenido del buffer en la posición indicada dentro del parámetro de entrada `Index`.

Parámetros

3. `byte Index`
   Índice del dato que se quiere recuperar del buffer.

3

```
dword GetDataLength()
```

Descripción general

Este método nos indica el tamaño del buffer de lectura de la última lectura correcta realizada.

Retorno

dword

Retorna la longitud de la cadena leída desde el escáner.

Parámetros

Este método no requiere parámetros de entrada.

4

```
dword GetStatus()
```

Descripción general

Este método sirve para obtener el estatus de lectura del escáner.

Retorno

dword

Retorna un valo que indica el estado en el que se encuentra el lector:

a. 1 → Código de barras leído correctamente.
b. 2 → Error de lectura de la etiqueta.
c. 4 → Buffer overrun, el tamaño del buffer interno del componente ha sido sobrepasado.
d. En caso de cualquier otro valor, se indica una combinación de flags (pueden estar activos varios a la vez), con la siguiente interpretación:

Bit 4 → Error de comunicaciones con el PLC, lo que implica un fallo en el protocolo de telegramas usado para comunicar el Gateway con el escáner. Es necesario revisar la configuración de Host Interface del escáner.

Bit 5 → Timeout de comunicación. El escáner no ha respondido a un telegrama de usuario en menos de 50 mseg.

Bit 6 → El escáner ha rechazado un telegrama de usuario mediante un telegrama NACK.

Bit 7 → Buffer Overrun en el Gateway que comunica el escáner con el bus.

Bit 8 → El interface reporta un error de lectura en el SCC (paridad, framing, etc.)

Parámetros

Este método no requiere parámetros de entrada.

5

```
void Reset()
```

Descripción general

Este método limpia el buffer de lectura de sus datos actuales y resincroniza la comunicación con el Gateway a través del bus. Debe invocarse en caso de que GetStatus nos devuelva algún estado de error.

Retorno

void

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

4.2.19 SICK: ESCÁNER PROFIBUS

4.2.19.1 Interface

Para poder realizar lecturas de etiquetas se ha desarrollado un componente que realiza el interface con la familia de escáner de Sick CLV vía Profibus. El Interface desarrollado funciona a través del cable serie. En las siguientes secciones explicaremos tanto la configuración del scanner como lo métodos expuestos por el componente.
Este componente soporta el Interface IScanner.

Los datos de creación del componente son los siguientes:

Class	SickPB
BUS IN	32 bytes
BUS OUT	32 bytes

El funcionamiento de este escáner es levemente diferente al del resto de la familia, dado que su sistema de comunicación se basa en telegramas enviados a través de la periferia del bus, no necesita de un hilo aparte para recoger las lecturas del scanner, sino que las va obteniendo de los telegramas. Si una lectura es buena se almacena en el buffer, si una lectura es mala se borra el buffer. El usuario debe consultar el status para decidir si los datos que se encuentran en el buffer corresponden a una lectura correcta.

Este componente sólo puede ser usado en un bus de campo ProfiBus.
Este componente soporta el Interface IScanner.

Listado de métodos:

1	bool ConfigScanner(byte Port, string Host)
2	byte GetData(byte Index)
3	dword GetDataLength()
4	dword GetStatus()
5	void Reset()

Descripción de los métodos:

1

```
bool ConfigScanner(byte Aux1,
                   string Aux2)
```

Descripción general

Este método debe invocarse en las rutinas de arranque, pero realmente no es necesario en este componente, que está listo para funcionar si se direcciona bien su posición en el bus, por lo tanto, sus parámetros no son usados, y la única razón de su existencia es para que el componente soporte el interface IScanner.

Retorno

`true`

Este método siempre retorna `true`.

Parámetros

3. `byte` Aux1
   Auxiliar.
4. `string` Aux2
   Auxiliar.

2

```
byte GetData(byte Index)
```

Descripción general

Cuando la lectura del escáner es correcta los datos leídos se almacenan en un buffer interno. Este método permite recuperar estos datos del mismo.

Retorno

`byte`

Retorna el valor del contenido del buffer en la posición indicada dentro del parámetro de entrada `Index`.

Parámetros

4. `byte` Index
   Índice del dato que se quiere recuperar del buffer.

3

```
dword GetDataLength()
```

Descripción general

Este método nos indica el tamaño del buffer de lectura de la última lectura correcta realizada.

Retorno

`dword`

Retorna la longitud de la cadena leída desde el escáner.

Parámetros

Este método no requiere parámetros de entrada.

4

```
dword GetStatus()
```

Descripción general

Este método sirve para obtener el estatus de lectura del escáner.

Retorno

dword

Retorna un valo que indica el estado en el que se encuentra el lector:

d. 1 → Código de barras leído correctamente.
e. 2 → Error de lectura de la etiqueta.
f. 4 → Buffer overrun, el tamaño del buffer interno del componente ha sido sobrepasado.
d. En caso de cualquier otro valor, se indica una combinación de flags (pueden estar activos varios a la vez), con la siguiente interpretación:
Bit 4 → Error de comunicaciones con el PLC, lo que implica un fallo en el protocolo de telegramas usado para comunicar el Gateway con el escáner. Es necesario revisar la configuración de Host Interface del escáner.
Bit 5 → Timeout de comunicación. El escáner no ha respondido a un telegrama de usuario en menos de 50 mseg.
Bit 6 → El escáner ha rechazado un telegrama de usuario mediante un telegrama NACK.
Bit 7 → Buffer Overrun en el Gateway que comunica el escáner con el bus.
Bit 8 → El interface reporta un error de lectura en el SCC (paridad, framing, etc.)

Parámetros

Este método no requiere parámetros de entrada.

5

```
void Reset()
```

Descripción general

Este método limpia el buffer de lectura de sus datos actuales y resincroniza la comunicación con el Gateway a través del bus. Debe invocarse en caso de que GetStatus nos devuelva algún estado de error.

Retorno

void

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

4.2.20 SICK: ESCÁNER PCP

4.2.20.1 Interface

Este componente esta deprecado, dado que el fabricante ha dejado de fabricar dicho elemento. Actualmente se usan escáneres TCP en su lugar.
Al igual que el anterior, también existe un componente que controla un Scanner Sick, pero con la particularidad de que este Scanner funciona usando mensajería PCP.
Para poder realizar lecturas de etiquetas se ha desarrollado un componente que realiza el interface con la familia de escáner de Sick CLV vía Profibus.

Class	SickClvPCP
BUS IN	2 bytes
BUS OUT	2 bytes

El funcionamiento del mismo es idéntico (desde el punto de vista del programador en Xana) al anterior. La diferencia se basa en cómo maneja se basa en un thread en background que recoge constantes lecturas del scanner, si una lectura es buena se almacena en el buffer, si una lectura es mala se borra el buffer. El usuario debe consultar el status para decidir si los datos que se encuentran en el buffer corresponden a una lectura correcta.

Listado de métodos:

1	bool ConfigScanner(byte Port, string Host)
2	byte GetData(byte Index)
3	dword GetStatus()
4	void Reset()
5	void Stop()
6	void Trigger()

Descripción de los métodos:

1

```
bool ConfigScanner(byte SlaveAddress)
```

Descripción general

Este método debe invocarse en las rutinas de arranque, permite abrir el puerto serie de comunicaciones para empezar a utilizar el escáner.

Retorno

`true`
Si la operación se ha realizado con éxito.
`false`
Si la operación NO se ha realizado con éxito.

Parámetros

1. `byte SlaveAddress`
   Dirección en el bus del escáner como esclavo de la red.

2

```
byte GetData(byte Index)
```

Descripción general

Cuando la lectura del escáner es correcta los datos leídos se almacenan en un buffer interno. Este método permite recuperar estos datos del mismo.

Retorno

`byte`
Retorna el valor del contenido del buffer en la posición indicada dentro del parámetro de entrada `Index`.

Parámetros

1. `byte` `Index`
   Índice del dato que se quiere recuperar del buffer.

3

```
dword GetStatus()
```

Descripción general

Este método sirve para obtener el estatus de lectura del escáner.

Retorno

`dword`
Retorna un valo que indica el estado en el que se encuentra el lector:

a. `1` → Código de barras leído correctamente.
b. `2` → Error de lectura de la etiqueta.
c. `3` → "Basura" en la comunicación.
d. `4` → Desbordamiento del buffer.

Parámetros

Este método no requiere parámetros de entrada.

4

```
void Reset()
```

Descripción general

Este método limpia el buffer de lectura de sus datos actuales y resincroniza la comunicación con el Gateway a través del bus. Debe invocarse en caso de que `GetStatus` nos devuelva algún estado de error.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

5

```
void Stop()
```

Descripción general

Este método detiene la lectura del escáner.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

6

```
void Trigger()
```

Descripción general

Dispara el trigger de lectura del escáner. Al recibirlo el escáner activa el haz de lectura e intenta leer códigos hasta que reciba un evento de parada mediante la invocación a `Stop`.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

4.2.21 LEUZE: ESCÁNER PROFIBUS LEUZE 504i

4.2.21.1 Introducción

El componente Leuze504i permite el control del scanner de etiquetas 504i de Leuze.

Este componente soporta el Interface IScanner.

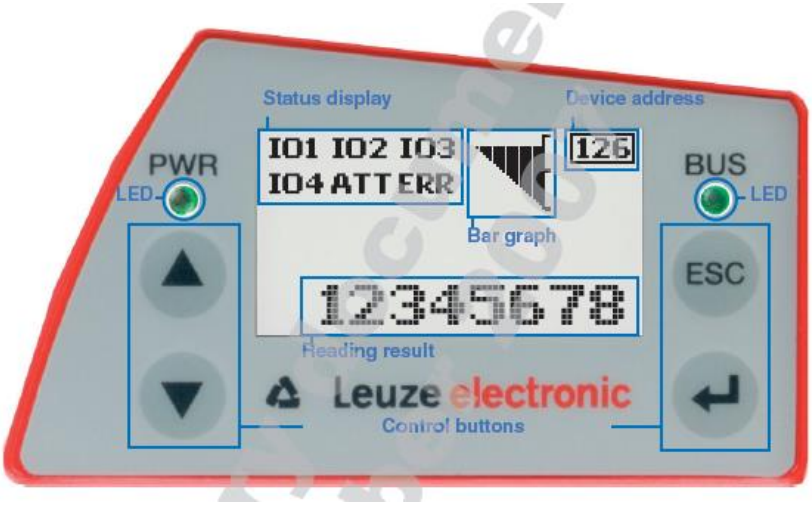
4.2.21.2 Configuración del escáner

La configuración del scanner se realiza desde la utilidad Sycon de Hilscher y desde el display, que a tal efecto tiene el scanner. Aunque este scanner tiene interface USB y dispone de una aplicación gráfica para su configuración, ésta solo tiene efecto cuando se utiliza en modo de transmisión serie, no Profibus, por lo que esta aplicación no será de utilidad en el caso de utilizar este componente.

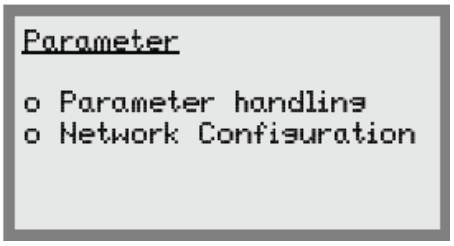
La configuración del escáner se realiza en dos partes: una desde el propio scanner y otra desde el programa Sycon.

4.2.21.2.1 Configuración desde el propio escáner

Lo primero que se tiene que realizar en este scanner es asignarle una dirección Profibus. Esto utilizando el display que incorpora el scanner:



Para asignarle la dirección tenemos que darle tensión al scanner. En ese momento el Led PWR empezará a lucir de forma intermitente. Esto ocurre durante el proceso de inicialización. En el momento en que el LED PWR quede fijo se podrá a empezar a trabajar con el escáner. Por defecto, el scanner trae deshabilitada la opción de cambio de parámetros, por lo que lo primero es cambiar esta opción de manera que podamos ponerle la dirección Profibus que deseemos. Para ello utilizaremos las flechas para cambiar entre las distintas pantallas, hasta encontrar la titulada como "*Parameter*":

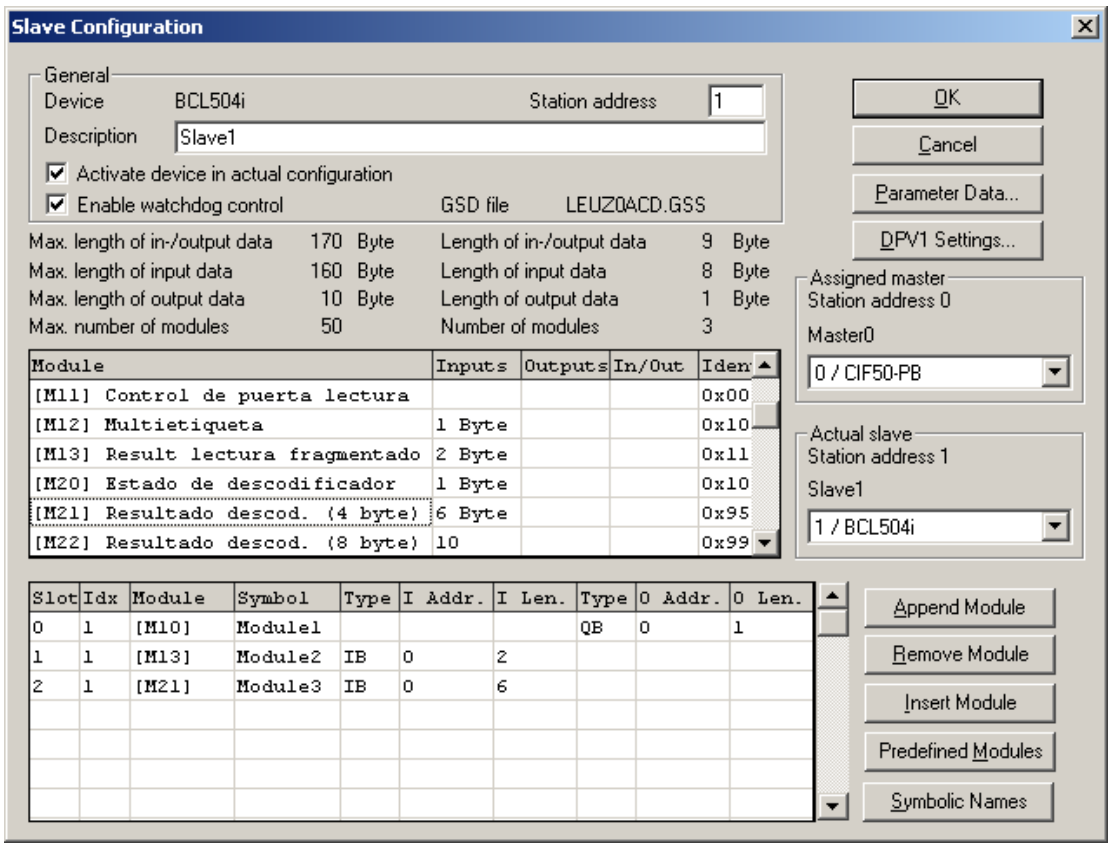


Desde esta pantallas, debemos primero habilitar el cambio de parámetros mediante la opción "*Parameter handling*" y después poner la dirección que queramos desde "*Network configuration*".

4.2.21.2.2 Configuración desde la aplicación Sycon

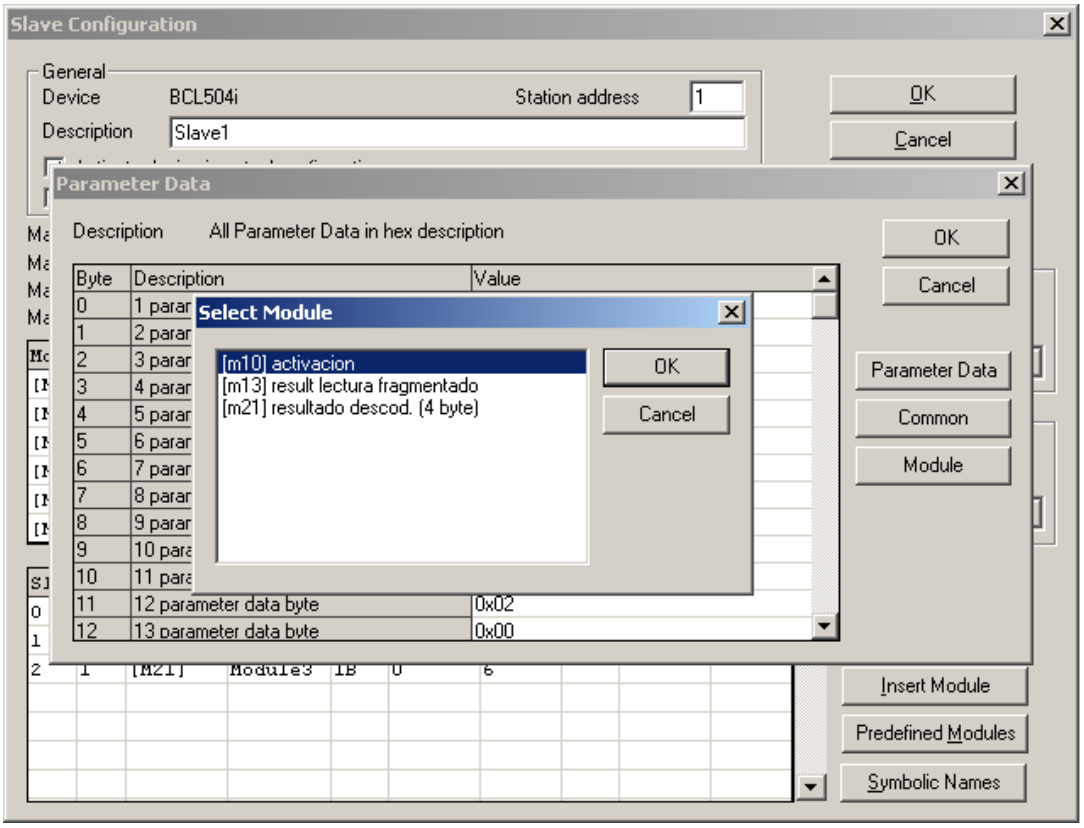
Una vez le hayamos puesto una dirección al scanner, podemos meterlo dentro del bus. Es importante que se pongan los siguientes módulos Y EN ESTE ORDEN durante la configuración del scanner en Sycon:

1. Módulo 10 – Activaciones
2. Módulo 13 – Resultado fragmentado
3. Módulo 21 – Resultado de 4 bytes



Se pueden añadir más módulos SIEMPRE QUE NO OCUPEN ESPACIO DE E/S. En el caso de añadir más módulos es responsabilidad del usuario su correcta configuración.

Una vez se tengan añadidos los módulos es necesario configurarlos correctamente. Para ello se pulsara el botón Parameter Data y en la pantalla que aparecerá el botón Module.



Aparecerá entonces un formulario para elegir el modulo a configurar.

4.2.21.2.2.1 Configuración del módulo de activación

En este módulo solo tendremos que configurar el modo en que se transmiten las etiquetas. En el caso de este componente ES IMPRESCINDIBLE que se configure a "Con Acknowledge". En este modo de funcionamiento, el scanner esperará antes de enviar otra etiqueta a que el componente le informe que puede hacerlo.

4.2.21.2.2.2 Configuración del módulo de lectura fragmentada

A fin de reducir el espacio de E/S y de hacer el componente los más genérico posible, la recepción de las etiquetas se realiza en varias etapas. De esta manera se pueden leer varios tipos distintos de etiquetas independientemente de su longitud. El único dato a configurar en este módulo es el tamaño del fragmento de dato, que tiene que ser OBLIGATORIAMENTE 4.

4.2.21.2.2.3 Configuración del módulo de resultado de 4 bytes

Este módulo no requiere configuración.

4.2.21.2.2.4 Configuración de los parámetros generales

Una vez se tengan configurados los módulos, es necesario configurar los parámetros generales del escáner. Esto se realiza pulsando en el botón "Parameter Data" desde el formulario de configuración del dispositivo y después pulsando en el

botón "Common". Aparecerá entonces una lista con los distintos parámetros a configurar. Realmente se trata de un conjunto de parámetros que se repite 4 veces y que permite definir los distintos tipos de códigos que se pueden leer. La configuración de estos parámetros es específica de cada aplicación, por lo que aquí simplemente vamos a indicar lo que es cada uno:

- *Tipo de código*: tipo de código a leer, por ejemplo EAN 128, 2/5 interleaved, etc.
- *Modo número dígitos*: este parámetro indica como se va a definir el número de dígitos del código. Si se pone a modo enumeración, los parámetros "numero de dígitos 1..5" son interpretados como una lista de tamaños que se podrán. Cualquier tamaño que no sea exactamente alguno de los definidos no será reconocido. Si se pone a modo rango, el parámetro "numero de dígitos 1" es tomado como el limite inferior y el parámetro "número de dígitos 2" como el limite superior de los posibles tamaños para ese código. Es por tanto mas flexible el modo rango, aunque también menos selectivo a la hora de leer códigos.
- *Numero de dígitos X*: dependiendo del parámetro anterior, el tamaño del código o el rango de su tamaño.
- El resto de parámetros se pueden dejar a su valor por defecto.

4.2.21.3 Interface

Los datos de creación del componente son los siguientes:

Class	Leuze504i
CAN IN	8 bytes
CAN OUT	1 bytes

Listado de métodos:

1	bool ConfigScanner(byte Aux1, string Aux2)
2	byte GetData(byte Index)
3	dword GetStatus()
4	void Reset()
5	void StartRead()

Descripción de los métodos:

1

```
bool ConfigScanner(byte Aux1,
                   string Aux2)
```

Descripción general

En este escáner no es necesario utilizar este método ni tiene ningún efecto. Esta disponible por motivos de compatibilidad.

Retorno

`true`

Este método siempre retorna `true`.

Parámetros

1. `byte` Aux1
   Auxiliar.
2. `string` Aux2
   Auxiliar.

2

```
byte GetData(byte Index)
```

Descripción general

Cuando la lectura del escáner es correcta los datos leídos se almacenan en un buffer interno. Este método permite recuperar estos datos del mismo.

Retorno

`byte`

Retorna el valor del contenido del buffer en la posición indicada dentro del parámetro de entrada `Index`.

Parámetros

1. `byte` Index
   Índice del dato que se quiere recuperar del buffer.

3

```
dword GetStatus()
```

Descripción general

Este método sirve para obtener el estatus de lectura del escáner.

Retorno

`dword`

Retorna un valor que indica el estado en el que se encuentra el lector:

a. `1` → Código de barras leído correctamente.
b. `2` → Error de lectura de la etiqueta.
c. `3` → Basura en el puerto de comunicaciones.
d. `4` → Desbordamiento del buffer.

Parámetros

Este método no requiere parámetros de entrada.

4

```
void Reset()
```

Descripción general

Este método limpia el buffer de lectura de sus datos actuales. Es recomendable realizar un `Reset` entre lecturas.

Retorno

void

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

5

```
void StartRead()
```

Descripción general

Este método inica la lectura en los escáner en los que sea necesario dispararla desde el programa de control.

Retorno

void

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

4.2.22 LEUZE: ESCÁNER PROFIBUS BCL34

4.2.22.1 Introducción

El componente LeuzeBCL34 permite el control del scanner de etiquetas BCL34 de Leuze.

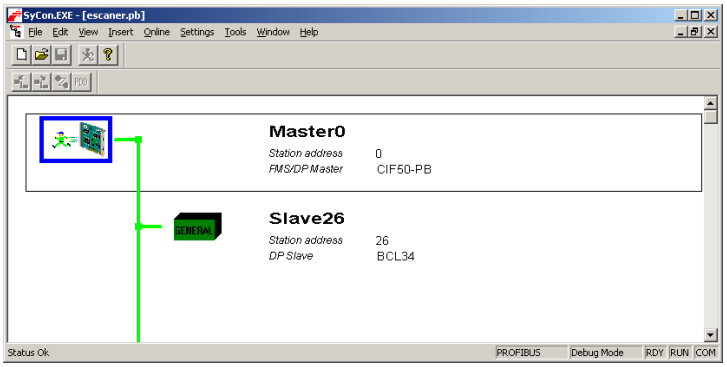
Este componente soporta el Interface IScanner.

4.2.22.2 Configuración del escáner

La configuración del scanner se realiza desde la utilidad Sycon de Hilscher.

4.2.22.2.1 Configuración desde la aplicación Sycon

Una vez le hayamos puesto una dirección al scanner, podemos meterlo dentro del bus. Si todo es correcto, una vez transferida la configuración a la tarjeta Hilscher (download), y posteriormente procediendo a realizar un Debug de la configuración con la comunicación iniciada, debería aparecernos todo en estado correcto (verde), tal y como se aprecia en la foto adjunta.



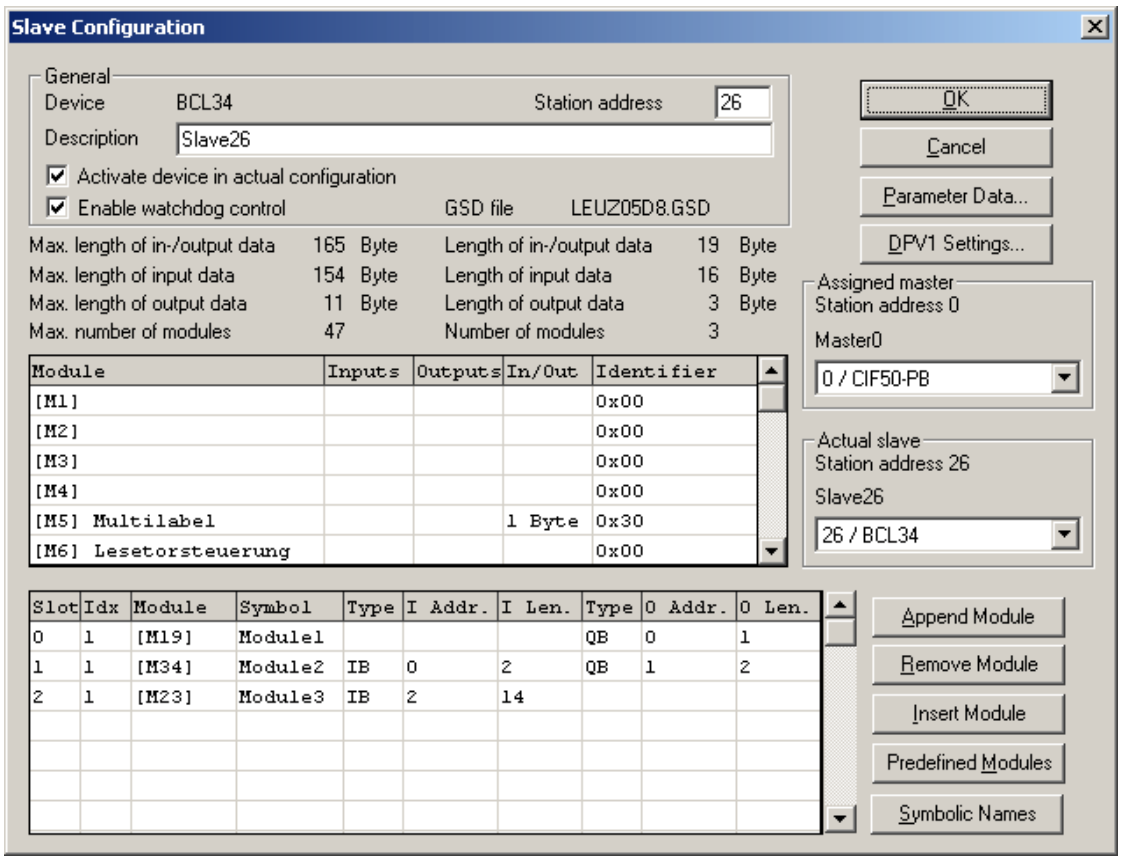
Hay que destacar, que disponemos de 2 tipos de configuraciones, en función de si pretendemos conseguir más velocidad a costa de sacrificar memoria, o viceversa.
Si optamos por la configuración básica, con poco consumo de memoria, pero con velocidades de lectura de 250 mseg. para un código completo de 12 bytes, es importante que se pongan los siguientes módulos Y EN ESTE ORDEN durante la configuración del scanner en Sycon:

- *Módulo 19* – Activaciones con ACK
- *Módulo 34* – Resultado de lectura fragmentado
- *Módulo 21* – Resultado de 4 bytes

Si por el contrario, optamos por una velocidad de lectura del orden de 20 ms, pero con mayor consumo de memoria, deberemos optar por los siguientes módulos, Y EN ESTE ORDEN:

- *Módulo 19* – Activaciones con ACK
- *Módulo 34* – Resultado de lectura fragmentado
- *Módulo 23* – Resultado de 4, 8, 12, 16, 20, 24 o 28 bytes.

Hay que destacar que el Módulo 23 puede tener múltiples configuraciones, ocupando desde los 4 hasta los 28 `bytes`, en función de la longitud del código de barras. A mayor longitud, mayor consumo de memoria. Si por algún motivo el código fuera mayor de 28 `bytes`, debemos tener en cuenta la fragmentación. Esto sólo afecta al apartado de velocidad, y por tanto dejaría de ser una componente genérica. La longitud del código la sabemos gracias a la posición 3 del módulo de entrada.



Hay que destacar que el módulo 23 puede configurarse con diferentes valores en función de los `bytes` necesarios para almacenar el código de barras, en concreto, permite los valores de 4, 8, 12, 16, 20, 24 y 28 `bytes`. En cualquiera de ellos puede darse la fragmentación, en caso de que el código sea mayor que el espacio final de almacenamiento.

Se pueden añadir más módulos SIEMPRE QUE NO OCUPEN ESPACIO DE E/S. En el caso de añadir más módulos es responsabilidad del usuario su correcta configuración.

Una vez se tengan añadidos los módulos es necesario configurarlos correctamente. Para ello se pulsará el botón Parameter Data y en la pantalla que aparecerá el botón Module.

4.2.22.2.1.1 Configuración del módulo de activación

En este módulo solo tendremos que configurar el modo en que se transmiten las etiquetas. En el caso de este componente ES IMPRESCINDIBLE que se configure a "Con Acknowledge". En este modo de funcionamiento, el scanner esperará antes de enviar otra etiqueta a que el componente le informe que puede hacerlo.

4.2.22.2.1.2 Configuración del módulo de lectura fragmentada

Para reducir el espacio en memoria de E/S y para conseguir un componente los más genérico posible, la recepción de las etiquetas se realiza en varias etapas. De esta forma se pueden leer varios tipos distintos de etiquetas independientemente de su longitud. El único parámetro configurable de este módulo es el tamaño del fragmento de dato, que OBLIGATORIAMENTE debe ser 4 en el caso de usar el *módulo 21*, o bien un valor mayor, en caso de usar el *módulo 23*.

4.2.22.2.1.3 Configuración del módulo de resultado de 4 bytes

Este módulo no requiere configuración.

4.2.22.2.1.4 Configuración de los parámetros generales

Una vez configurados los módulos, se deben configurar los parámetros generales del scanner. Esto paso se realiza pulsando en el botón "*Parameter Data*" desde el formulario de configuración del dispositivo y posteriormente pulsando en el botón "*Common".* Acto seguido, nos aparecerá una lista con los distintos parámetros configurables. Dichos parámetros se repiten 4 veces y permiten definir los distintos tipos de códigos que se pueden leer. La configuración de estos parámetros es específica de cada aplicación, por lo que simplemente vamos a indicar lo que es cada uno:

- Tipo de código: tipo de código a leer, por ejemplo EAN 128, 2/5 interleaved, etc.
- Modo número dígitos: este parámetro indica como se va a definir el número de dígitos del código. Si se pone a modo enumeración, los parámetros "numero de dígitos 1..5" son interpretados como una lista de tamaños que se pondrán. Cualquier tamaño que no sea exactamente alguno de los definidos no será reconocido. Si se pone a modo rango, el parámetro "numero de dígitos 1" es tomado como el limite inferior y el parámetro "número de dígitos 2" como el limite superior de los posibles tamaños para ese código. Es por tanto mas flexible el modo rango, aunque también menos selectivo a la hora de leer códigos.
- Número de dígitos X: dependiendo del parámetro anterior, el tamaño del código o el rango de su tamaño.

El resto de parámetros se pueden dejar a su valor por defecto.

4.2.22.3 Interface

Los datos de creación del componente son los siguientes:

Este componente es compatible con el interface IScanner

Class	LeuzeBCL34
BUS IN	8 bytes
BUS OUT	1 bytes

Class	LeuzeBCL34
BUS IN	12 bytes
BUS OUT	1 bytes

Listado de métodos:

1	`bool ConfigScanner(byte Aux1, string Aux2)`
2	`byte GetData(byte Index)`
3	`dword GetStatus()`
4	`void Reset()`
5	`void StartRead()`

Descripción de los métodos:

1

```
bool ConfigScanner(byte Aux1,
                   string Aux2)
```

Descripción general

En este escáner no es necesario utilizar este método ni tiene ningún efecto. Esta disponible por motivos de compatibilidad.

Retorno

`true`

Este método siempre retorna `true`.

Parámetros

1. `byte` Aux1
   Auxiliar.
2. `string` Aux2
   Auxiliar.

2

```
byte GetData(byte Index)
```

Descripción general

Cuando la lectura del escáner es correcta los datos leídos se almacenan en un buffer interno. Este método permite recuperar estos datos del mismo.

Retorno

`byte`

Retorna el valor del contenido del buffer en la posición indicada dentro del parámetro de entrada `Index`.

Parámetros

2. `byte` Index
   Índice del dato que se quiere recuperar del buffer.

3

```
dword GetStatus()
```

Descripción general

Este método sirve para obtener el estatus de lectura del escáner.

Retorno

`dword`

Retorna un valor que indica el estado en el que se encuentra el lector:

a. `1` → Código de barras leído correctamente (READ_OK).
b. `2` → Error de lectura de la etiqueta (READ_ERROR).
c. `3` → Basura en el puerto de comunicaciones (TRASH).
d. `4` → Desbordamiento del buffer (OVERFLOW).

Parámetros

Este método no requiere parámetros de entrada.

4

```
void Reset()
```

Descripción general

Este método limpia el buffer de lectura de sus datos actuales. Es recomendable realizar un `Reset` entre lecturas.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

5

```
void StartRead()
```

Descripción general

Este método inica la lectura en los escáner en los que sea necesario dispararla desde el programa de control.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

4.2.23 BÁSCULA DE PESAJE MICROGRAM

4.2.23.1 Introducción

Para poder realizar control de peso en ciertas instalaciones, es necesario incluir un sistema de bascula que nos de una lectura sobre el peso de una unidad de almacenaje. A estos efectos se ha trabajado con la báscula modelo IE21 de Microgram, a fin de desarrollar un interface que funciona a través del puerto serie, y que permite una comunicación con un elemento de estas características, a fin de obtener la información señalada al inicio de este párrafo.

4.2.23.2 Elementos hardware

4.2.23.2.1 Modos de funcionamiento de la báscula

La bascula IE21 tiene 2 modos de funcionamiento: por petición (modo por defecto) o automático. En el modo por petición, es necesario que el dispositivo de control envíe una señal por el puerto serie cada vez que se quieran obtener los datos de una pesada. Esta tarea es realizada de forma automática por el componente, sin necesidad de intervención por el usuario. En el modo automático, la bascula transmite, cada 100 mseg., los datos de su estado y ultima pesada por el puerto serie. Es posible seleccionar con que modo se va a trabajar usando el método de configuración que proporciona el componente.

4.2.23.2.2 Comunicación vía serie

Para la conexión de la báscula con el dispositivo host necesitaremos un cable serie, con un conector hembra en uno de los extremos, que conectaremos al PC. En el otro extremo deberemos conectar los hilos 2, 3, 5, 7 y 8 según el siguiente esquema:

IE21	Descripción	PC	Descripción
2	RxD	3	TxD
3	TxD	2	RxD
5	GND	5	GND
7	RTS	8	CTS
8	CTS	7	RTS

Según el fabricante, también es necesario que las patillas 4 y 6 del puerto serie del PC se encuentren unidas, pero esto puede no ser necesario.

Para configurar la comunicación entre los dos elementos, el fabricante especifica una serie de restricciones en cuanto a la configuración de puerto serie:

- Velocidad de transmisión puede ser 1200 o 9600 baudios.
- Sin paridad (n) o con paridad impar (o).
- 7 bits de datos
- 1 ó 2 bits de stop, según la paridad sea ninguna o impar.

4.2.23.3 Interface

Class	BasculaIE21
BUS IN	0 bytes

BUS OUT 0 bytes

Listado de métodos:

1	`void BeginWaitForData()`

2	`bool Config(word Port,`
	`  string Parameter,`
	`  bool Mode,`
	`  byte Address)`

3	`bool DataAdquired()`
4	`void FinishWaitForData()`
5	`dword GetLastError()`
6	`byte GetStatus()`
7	`bool GetWeight(dword &Value)`
8	`void ResetStatus()`
9	`void SetDivisor(dword Divisor1,` `  dword Divisor2)`

Descripción de los métodos:

1

```
void BeginWaitForData()
```

Descripción general

Normalmente, el componente no se encuentra escuchando el puerto serie, es necesario una llamada a este método para hacer que comienze el ciclo de petición – escucha para obtener datos de la báscula.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

2

```
bool Config(word Port,
            string Parameter,
            bool Mode,
            byte Address)
```

Descripción general

Este método debe invocarse en las rutinas de arranque, permite abrir el puerto serie de comunicaciones para empezar a utilizar la báscula.

Retorno

`true`

Si la operación se ha realizado con éxito y la báscula ha quedado correctamente configurada y lista para su uso.

`false`
Si la operación de configuración no se ha completado correctamente.

Parámetros

1. `word` `Port`
   Puerto COM que se desea inicializar (normalmente de 1 a 4).
2. `string` `Parameter`
   Configuración de la velocidad y datos de transmisión. Para la configuración de la báscula cabe elegir entre varias alternativas, pero siempre teniendo en cuenta las limitaciones expuestas en el apartado de configuración de la comunicación con la báscula, por ejemplo "`1200,o,7,1`"
3. `bool` `Mode`
   Permite seleccionar si se quiere que la báscula trabaje en modo automático (`false`) o por petición del sistema de control (`true`)
4. `byte` `Address`
   Dirección de la báscula que por defecto es 0 y que debe de discriminar cuando la petición que le llega por el puerto va o no dirigida a ella.

3

```
bool DataAdquired()
```

Descripción general

Este método sirve para indicarnos si se ha obtenido alguna nueva pesada desde que se invocó al método `BeginDataAdquire`. Notese que si la comunicación está detenida (bien porque aún no se haya llamado a `BeginDataAdquire` o porque se haya invocado a `FinishDataAdquire`), este método revolverá siempre `false`.

Retorno

`true`
Se ha obtenido una pesada válida desde la última invocación al método `BeginDataAdquire.`
`false`
La comunicación está pausada, o no se ha obtenido aún un valor de peso sin error.

Parámetros

Este método no requiere parámetros de entrada.

4

```
void FinishWaitForData()
```

Descripción general

Una vez que se obtiene un peso estable, ya no es necesario mantener el puerto serie abierto, ni que el componente este escuchando – requiriendo continuamente el envío de datos. Al invocar este método, se realiza una pausa en el hilo de comunicación entre el componente y la báscula, lo cual sirve para no consumir recursos innecesarios y evita también fallos de comunicación.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

5

```
dword GetLastError()
```

Descripción general

Este método nos proporciona información sobre el último error que se recibo al intentar comunicarse con la báscula. Es decir, la última vez que la báscula nos ha respondido a la petición de lectura con un NACK. En general, todos los errores que se produzcan indican un problema de comunicaciones, que nos obligará a revisar la configuración del puerto o a revisar el cable serie que lo conecta a la báscula.

Retorno

dword

Retorna el valor que indica la causa del error. Sus posibles valores son:

a. 0x31 → Error de recepcion
b. 0x32 →Falta ETX
c. 0x33 → Caracter no permitido
d. 0x34 → ENQ en posicion incorrecta
e. 0x35 → Longitud de DATO incongruente
f. 0x36 → Buffer lleno
g. 0x37 → Codigo no reconocido
h. 0x38 → Se esperaban más datos en el mensaje
i. 0x39 → Sobran datos en el mensaje
j. 0x41 → Error en la longitud de mensaje
k. 0x42 → Comando imposible de ejecutar
l. 0x43 → Valor no correcto
m. Cualquier otro valor indica un error desconocido.

Parámetros

Este método no requiere parámetros de entrada.

6

```
byte GetStatus()
```

Descripción general

Este método sirve para obtener el estado de comunicación con la báscula.

Retorno

dword

Retorna un valor que indica el estado en el que se encuentra la báscula:
0 → La báscula no se encuentra en funcionamiento o aún no ha leído nada.
1 → Peso leído correctamente.
2 → Error en la lectura del peso.
3 → "Basura" en el puerto de comunicaciones.
4 → Desbordamiento del buffer.
5 → Fallo en la interpretación del mensaje de respuesta de la báscula.
Probablemente existe algún telegrama corrupto.

Parámetros

Este método no requiere parámetros de entrada.

7

```
bool GetWeight(dword &Value)
```

Descripción general

Cuando la lectura de la báscula es correcta, el peso obtenido se puede recoger mediante este método.

Retorno

`true`
Si se ha adquirido algún peso sin ningún error.
`false`
Aún no se ha podido obtener ninguna lectura del peso.

Parámetros

1. `dword` `Value`
   Variable donde se almacenará el último valor guardado por el componente como peso leído.

8

```
void ResetStatus()
```

Descripción general

Este método pone el flag de estado de la báscula al valor 0.

Retorno

`void`
Este método no retorna nada.

Parámetros

Este método no requiere parámetros de entrada.

9

```
void SetDivisor(dword Divisor1,
                dword Divisor2)
```

Descripción general

Este método sirve para establecer uno de los parametros internos de la báscula. Según el manual del fabricante, la lectura del peso correcta se obtiene aplicando una formula matemática al valor leido, en la que hay una constante cuyo valor por defecto es 30,0. Dado que esto podria variar en el futuro, este método permitiria al componente funcionar con otra versión de la báscula sin tener que ser recompilado. Como dicho valor tiene componente decimal, y en xana no se permiten las variables de tipo flotantes, el valor final se pasa como el resultado de la division de los dos parámetros pasados como parámetros.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `dword` `Divisor1`
   Define el divisor 1 necesario para poder establecer el número en coma flotante

que se obtiene mediante la operación `Divisor1/Divisor2` y que será el que se use como divisor del resultado de la báscula.
2. `dword Divisor2`
Define el divisor 2 necesario para poder establecer el número en coma flotante que se obtiene mediante la operación `Divisor1/Divisor2` y que será el que se use como divisor del resultado de la báscula.

4.2.24 LECTOR DE TAGS KL6001_RFM12 (Leuze y Beckhoff)

4.2.24.1 Introducción

El propósito de este apartado es explicar como se realiza la programación de este componente que en realidad consiste en dos componentes hardware agrupados: por un lado la antena lectora de tags ML_12 de Leuze y por el otro el modulo de Beckhoff KL6001 de comunicaciones serie sobre Profibus.

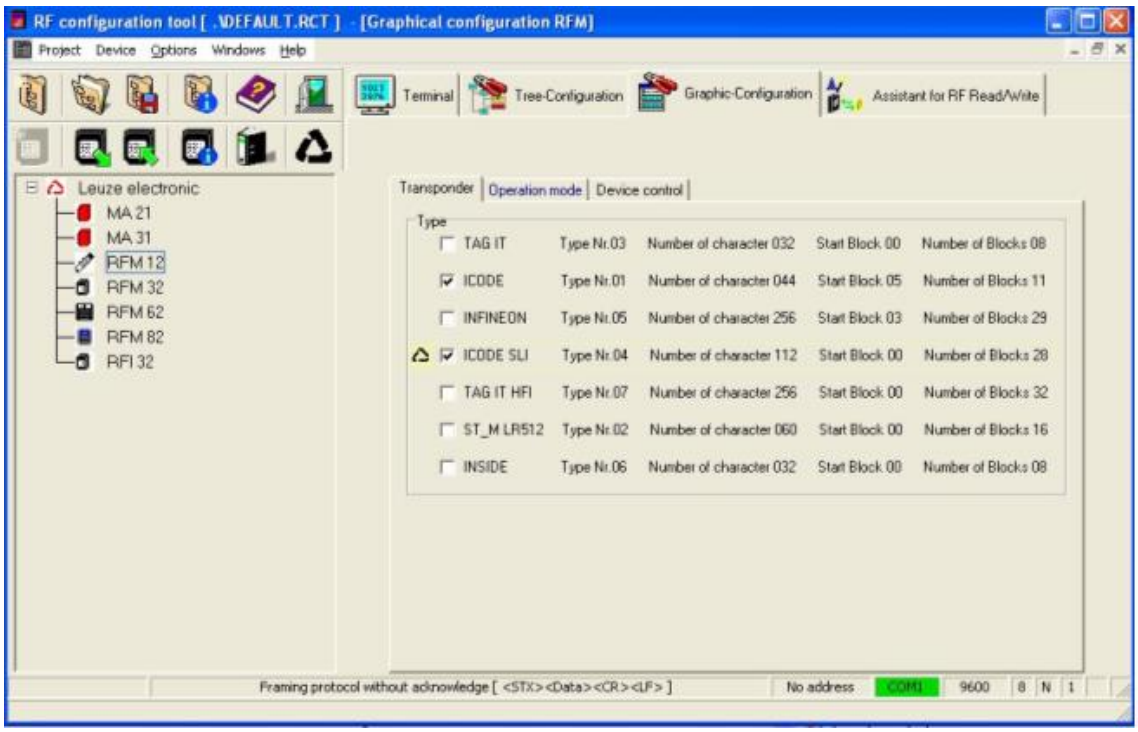
4.2.24.2 Características técnicas y configuraciones

La antena lectora de Tags ML_12 de Leuze es un sistema de lectura de tags magnéticos que inicialmente funciona sobre puerto serie, enviando y recibiendo datos por un puerto de este tipo, y actuando básicamente mediante comandos de control que se envían y reciben a través de este puerto. Por otro lado, el componente KL6001 de Beckhoff realiza el trabajo de simular la comunicación de un puerto serie implementándola sobre un bus Profibus. Es decir: implementa un buffer de transmisión/recepción de datos con control del mismo sobre un puerto serie normal. Su propósito principal es el de poder permitir a un sistema de control gobernar un puerto serie desde el propio bus de control.

Los dos componentes por separado deben ser configurados por separado mediante sus aplicaciones correspondientes. Esta configuración debe realizarse de forma cuidadosa y adherirse por completo a lo que en este documento se relata.

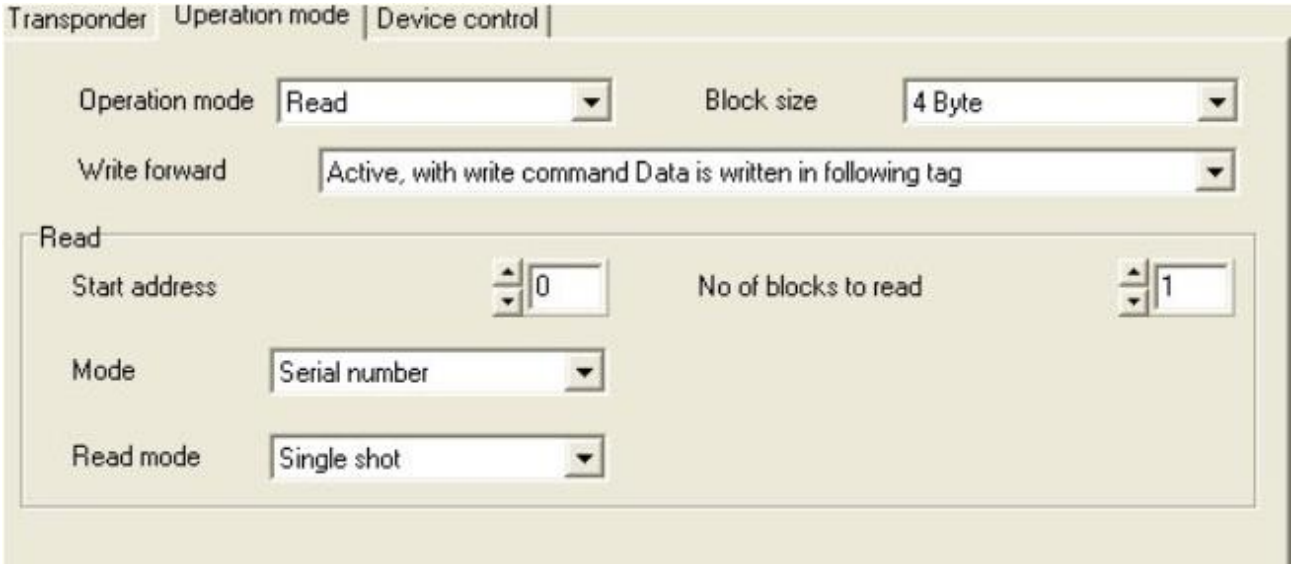
4.2.24.2.1 Configuración de la antena ML_12 de Leuze

El programa para configurar este elemento es el "RF Configuration Tool" de Leuze. Previamente a su uso deberemos conectar la antena por un puerto serie al PC desde el que lanzamos la aplicación. Veremos una pantalla similar a la que se muestra en la siguiente ilustración:

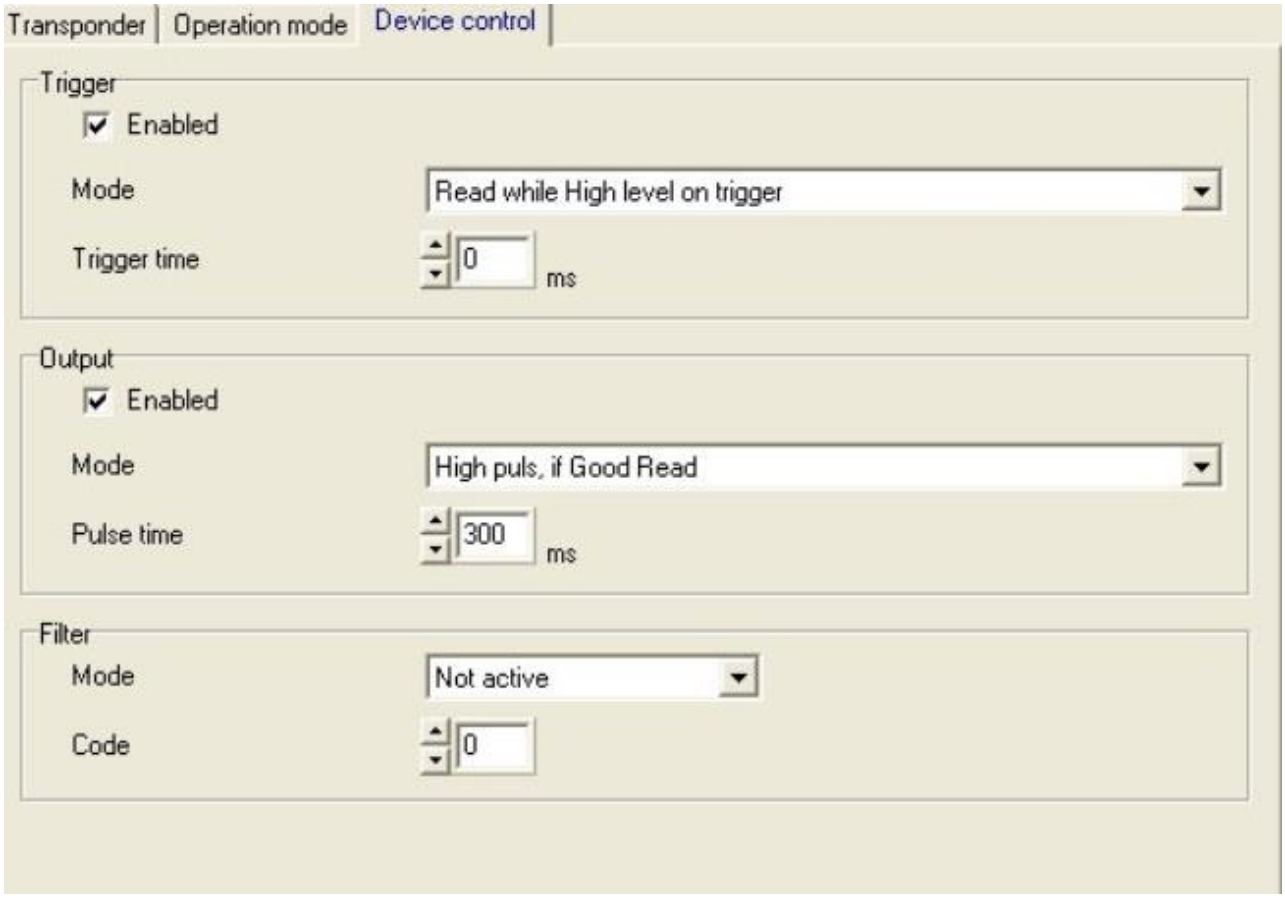


Debemos seleccionar el modelo de la antena en el árbol de la izquierda antes de poder proceder a su configuración. Una vez seleccionado, la barra inferior de la aplicación nos mostrara la información sobre la velocidad de conexión y estado de la misma (siempre será a 9600bps). En la solapa "Graphic Configuration" encontraremos 3 nuevas solapas que son las que finalmente nos permiten alterar el comportamiento de la antena. En el caso que nos ocupa, este componente fue desarrollado originalmente para soportar transponders (tags) de los modelos TFM05 1105.210. Dichos transponders usan el código de tag ICODE SLI (código 04), por lo que hay que asegurarse, como muestra la ilustración anterior), que dicho tipo de transponder esta seleccionado en la configuración de la antena.

El siguiente tag hace referencia al modo de operación en que actuara la antena. Veámoslo en la siguiente imagen:



Estos son los valores por defecto, así que no será habitual tener que tocar nada en esta pantalla.
La última solapa permite cambiar parámetros de control del dispositivo, y contiene valores similares a los de la siguiente ilustración:

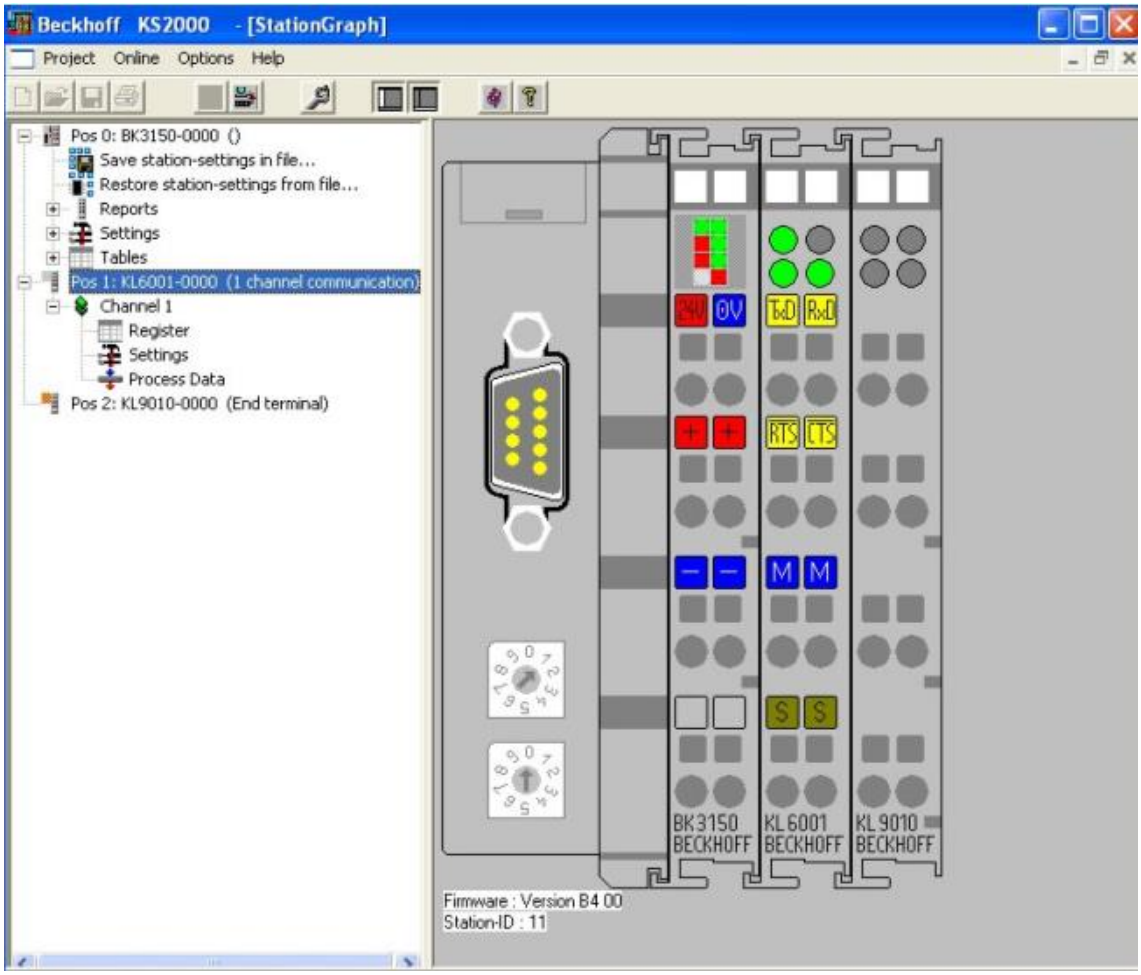


Como la anterior, tampoco es necesario cambiar nada en esta solapa, ya que los valores por defecto permiten el uso del componente.

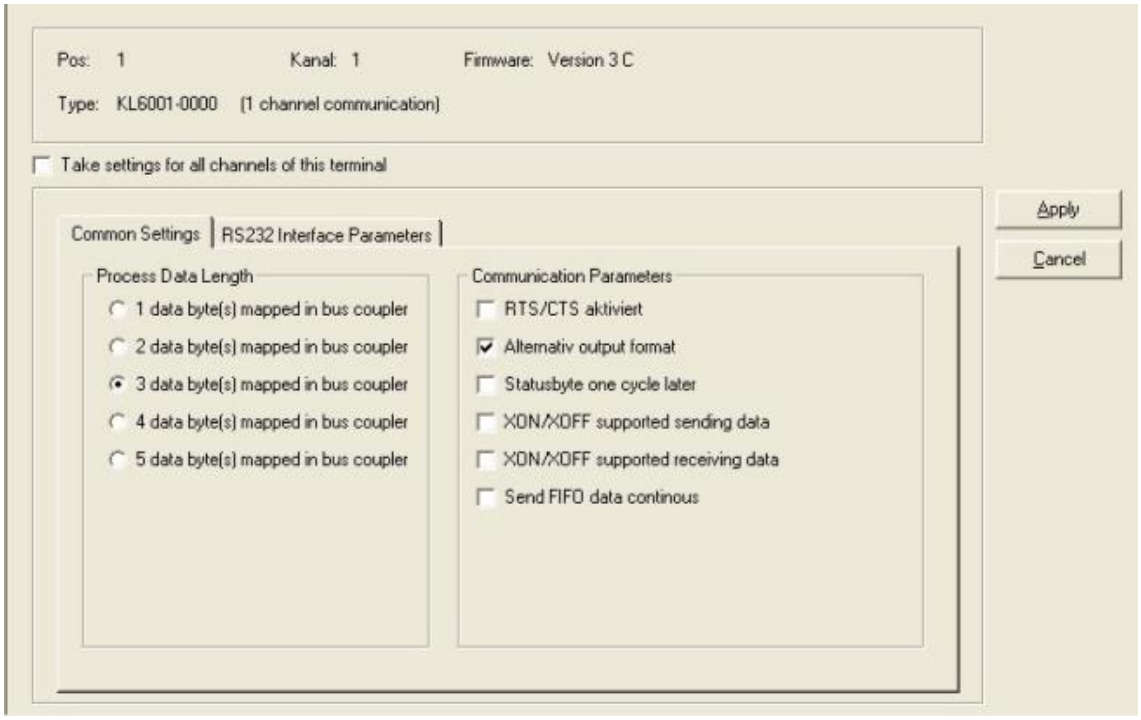
Una vez establecidos los valores apropiados, solo es necesario usar la opción del menú "Device → Transmit Parameters" para guardar la configuración en el dispositivo y tener este listo para su uso con el componente Galileo.

4.2.24.2.2 Configuración del Modulo KL6001 de Beckhoff

El software para configurar este elemento es el "Beckhoff KS2000" del fabricante. Para poder usarlo es necesario conectar un cable de programación del modulo BK3150 (ó 31XX, modulo al que va acoplado el KL6001 en el bus) con un puerto serie del PC desde el que se ejecuta el software de configuración. Una vez lanzado el software, seleccionaremos la opción "Online → Login" para intentar conectarnos con el modulo. Si la comunicación es posible, llegaremos a una pantalla como la que mostramos a continuación (es posible que recibamos algunos avisos durante el proceso).



La parte que nos interesa de la configuración se encuentra en la rama del árbol de la izquierda de reza "Settings", y que nos lleva a la siguiente pantalla:



Debemos asegurarnos que en "common settings" se mapean 3 data `bytes` como "Process Data Length", que **no** se usan parámetros de ningún tipo en la comunicación y que se usa el formato de salida alternativo.

En cuanto a la solapa de parámetros del interfaz RS232, debemos escoger los siguientes:

Estos parámetros deben ser exactamente 9600 bps, 8 bits de datos, 1 bit de stop y sin paridad. Tras comprobar la configuración, solo tendremos que pulsar el botón

"Apply Settings" para transferir la configuración al dispositivo y tenerlo listo para trabajar con la antena ML_12.

4.2.24.2.3 Cableado

La configuración de cableado es mediante un simple cable serie de 3 hilos cruzado, que se mapean al KL6001 según la siguiente figura:

Los terminales 1, 3 y 5 vienen marcados por las etiquetas TxD, M y RxD respectivamente, y serán los que tendremos que conectar a las señales RxD, M y TxD del puerto serie de la antena. Una vez realizado dicho cableado, tendremos el sistema listo para ser configurado y usado desde Galileo.

4.2.24.3 Interface

Class	KL6001_RFM12
BUS IN	6 bytes
BUS OUT	6 bytes

Listado de métodos:

1	`bool Busy()`
2	`dword GetState()`
3	`byte GetValue(byte BlockNumber, byte BlockSize, byte Index)`
4	`void LoadTag(byte BlockIndex, byte BlockSize, byte BlocksNumber, byte TagType)`
5	`void Putvalue(byte BlockNumber, byte BlockSize, byte Index, byte Value)`
6	`void Reset()`
7	`void SaveTag(byte BlockIndex, byte BlockSize, byte BlocksNumber, byte TagType)`
8	`void SetTimeOut(dword Timeout)`

Descripción de los métodos:

1

```
bool Busy()
```

Descripción general

Indica si hay comandos en la pila o no, esto indica que el objeto tiene aún comandos por despachar.

Retorno

`true`
Hay comandos pendientes en la pila.
`true`
NO hay comandos pendientes en la pila para despachar.

Parámetros

Este método no requiere parámetros de entrada.

2

```
dword GetState()
```

Descripción general

Indica el estado de las operaciones en la antena.

Retorno

`dword`
Retorna el estado de las operaciones que se han de realizar:

a. `0` → Operación pendiente o antena en reposo (sin realizar ninguna función). Este es el estado en que debería estar tras un comando de reset.
b. `1` → Lectura realizada con éxito.
c. `2` → Escritura realizada con éxito.
d. Cualquier otro valor distinto de los anteriores indica una condición de error (por ej.: un `8` indicaría error de timeout en una operación).

Parámetros

Este método no requiere parámetros de entrada.

3

```
byte GetValue(byte BlockNumber,
              byte BlockSize,
              byte Index)
```

Descripción general

Este método devuelve el valor de una celda de memoria del tag que previamente debe haber sido leída mediante un método LoadTag. Ha de tenerse en cuenta que esto solo obtiene el valor del buffer interno de memoria que se ve entre operaciones de lectura y escritura, con lo que dicho valor no es actualizado hasta que se produzca otra operación de lectura/escritura con el transponder.

Retorno

`byte`
Valor de la celda de memoria indicada.

Parámetros

1. byte BlockNumber
Indica el número de bloque del transponder.
2. byte BlockSize
Indica el tamaño del bloque del transponder.
3. byte Index
Índice dentro del bloque cuyo valor queremos obtener.

4

```
void LoadTag(byte BlockIndex,
             byte BlockSize,
             byte BlocksNumber,
             byte TagType)
```

Descripción general

Este método permite cargar los contenidos de un bloque de memoria desde transponder que se encuentra en el rango de acción de la antena.

Retorno

void
El método no retorna parámetros. El sistema de control deberá chequear el componente mediante el método GetStatus de forma periódica hasta obtener un código de operación completada.

Parámetros

1. byte BlockIndex
Índice del primer bloque a salvar.
2. byte BlockSize
Indica el tamaño del bloque (bytes) del transponder.
3. byte BlocksNumber
Número de bloques a salvar.
4. byte TagType
Según el transponder a usar tendrá un código u otro (por ejemplo 4 es para el modelo TFM05 1105.210).

5

```
void Putvalue(byte BlockNumber,
              byte BlockSize,
              byte Index,
              byte Value)
```

Descripción general

Este método establece el valor de una celda de memoria del tag a un determinado valor. Ha de tenerse en cuenta que esto solo modifica el valor del buffer interno de memoria que se ve entre operaciones de lectura y escritura, con lo que dicho valor no es salvado al transponder hasta que se efectúa una operación de SaveTag.

Retorno

void
Este método no retorna ningún valor.

Parámetros

1. byte BlockNumber

Indica el número de bloque del transponder.

2. `byte BlockSize`

Indica el tamaño del bloque del transponder.

3. `byte Index`

Índice dentro del bloque cuyo valor queremos modificar.

4. `byte Value`

Valor que se desea asignar a la celda de memoria.

6

```
void Reset()
```

Descripción general

Este método permite reinicializar el funcionamiento del componente, así como reinicializar la comunicación con la antena y el modulo de bus. Debe ser llamado, por ejemplo, tras una operación con error.

Retorno

`void`

El método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

7

```
void SaveTag(byte BlockIndex,
             byte BlockSize,
             byte BlocksNumber,
             byte TagType)
```

Descripción general

Este método permite salvar los contenidos de un bloque de memoria al transponder que se encuentra en el rango de acción de la antena (siempre que este sea de tipo escritura).

Retorno

`void`

El método no retorna parámetros. El sistema de control deberá chequear el componente mediante el método `GetStatus` de forma periódica hasta obtener un código de operación completada.

Parámetros

1. `byte BlockIndex`

Índice del primer bloque a salvar.

2. `byte BlockSize`

Indica el tamaño del bloque (bytes) del transponder.

3. `byte BlocksNumber`

Número de bloques a salvar.

4. `byte TagType`

Según el transponder a usar tendrá un código u otro (por ejemplo 4 es para el modelo TFM05 1105.210).

8

```
void SetTimeOut(dword Timeout)
```

Descripción general

Este método establece el tiempo máximo, en milisegundos, que una operación puede estar realizándose sin respuesta antes de que el componente la aborte y señale un error de tiempo de espera agotado (código 8).

Retorno

`void`

El método no retorna ningún valor.

Parámetros

1. `dword Timeout`
   Tiempo en milisegundos de timeout para las operaciones de lectura, escritura y reset.

4.2.25 LECTOR DE TAGS KL6001_UDP1 (Beckhoff y Leuze)

4.2.25.1 Introducción

El propósito de este apartado es explicar como se realiza la programación de este componente que en realidad consiste en dos componentes hardware agrupados: Por un lado la antena lectora de tags UDP1 de Leuze y por el otro el modulo de Beckhoff KL6001 de comunicaciones serie sobre Profibus.

4.2.25.2 Caracteristicas técnicas

La antena lectora de Tags UDP1 de Leuze es un sistema de lectura de tags magnéticos que inicialmente funciona sobre puerto serie, enviando y recibiendo datos por un puerto de este tipo, y actuando básicamente mediante comandos de control que se envían y reciben a través de este puerto. Por otro lado, el componente KL6001 de Beckhoff realiza el trabajo de simular la comunicación de un puerto serie implementándola sobre un bus Profibus. Es decir: implementa un buffer de transmisión/recepción de datos con control del mismo sobre un puerto serie normal. Su propósito principal es el de poder permitir a un sistema de control gobernar un puerto serie desde el propio bus de control.

Los dos componentes por separado deben ser configurados por separado mediante sus aplicaciones correspondientes.

Por el momento, esta antena aún está en desarrollo y algunas funcionalidades que no están disponibles. Actualmente no es posible enviar comando para almacenar información en los Tags ni tampoco recuperar información almacenada en ellos. **S***olo es posible leer su etiqueta identificativa.*

4.2.25.3 Interface

Class	KL6001_UDP1

BUS IN	6 bytes
BUS OUT	6 bytes

Listado de métodos:

1	`bool Busy()`
2	`void ClearBuffer()`
3	`byte GetId(word Index)`
4	`word GetIdLen()`
5	`dword GetState()`
6	`byte GetValue(byte BlockNumber, byte BlockSize, byte Index)`
7	`void LoadId()`
8	`void LoadTag(byte BlockIndex, byte BlockSize, byte BlocksNumber, byte TagType)`
9	`void Putvalue(byte BlockNumber, byte BlockSize, byte Index, byte Value)`
10	`void Reset()`
11	`void SaveTag(byte BlockIndex, byte BlockSize, byte BlocksNumber, byte TagType)`
12	`void SetTimeOut(dword Timeout)`

Descripción de los métodos:

1

```
bool Busy()
```

Descripción general

Indica si hay comandos en la pila o no, esto indica que el objeto tiene aún comandos por despachar.

Retorno

`true`
Hay comandos pendientes en la pila.
`true`
NO hay comandos pendientes en la pila para despachar.

Parámetros

Este método no requiere parámetros de entrada.

2

```
void ClearBuffer()
```

Descripción general

Borra el buffer interno.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

3

```
byte GetId(word Index)
```

Descripción general

Devuelve el carácter *index* del codigo almacenado en buffer interno.

Retorno

`byte`
Carácter especificado.

Parámetros

1. `word` `Index`
   Índice del carácter a devolver.

4

```
word GetIdLen()
```

Descripción general

Devuelve la longitud del codigo cargado en buffer interno.

Retorno

`word`
Tamaño en caracteres del código cargado.

Parámetros

Este método no requiere parámetros de entrada.

5

```
dword GetState()
```

Descripción general

Indica el estado de las operaciones en la antena.

Retorno

`dword`
Retorna el estado de las operaciones que se han de realizar:

a. `0` → Operación pendiente o antena en reposo (sin realizar ninguna función). Este es el estado en que debería estar tras un comando de reset.
b. `1` → Lectura realizada con éxito.

c. 2 → Escritura realizada con éxito.
d. Cualquier otro valor distinto de los anteriores indica una condición de error (por ej.: un 8 indicaría error de timeout en una operación).

Parámetros

Este método no requiere parámetros de entrada.

6

```
byte GetValue(byte BlockNumber,
              byte BlockSize,
              byte Index)
```

Descripción general

Este método devuelve el valor de una celda de memoria del tag que previamente debe haber sido leída mediante un método LoadTag. Ha de tenerse en cuenta que esto solo obtiene el valor del buffer interno de memoria que se ve entre operaciones de lectura y escritura, con lo que dicho valor no es actualizado hasta que se produzca otra operación de lectura/escritura con el transponder.

Retorno

byte
Valor de la celda de memoria indicada.

Parámetros

1. byte BlockNumber
   Indica el número de bloque del transponder.
2. byte BlockSize
   Indica el tamaño del bloque del transponder.
3. byte Index
   Índice dentro del bloque cuyo valor queremos obtener.

7

```
void LoadId()
```

Descripción general

Carga el valor del identificador del Tag en un buffer interno.

Retorno

void
El método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

8

```
void LoadTag(byte BlockIndex,
             byte BlockSize,
             byte BlocksNumber,
             byte TagType)
```

Descripción general

Este método permite cargar los contenidos de un bloque de memoria desde transponder que se encuentra en el rango de acción de la antena.

Retorno

`void`

El método no retorna parámetros. El sistema de control deberá chequear el componente mediante el método `GetStatus` de forma periódica hasta obtener un código de operación completada.

Parámetros

1. `byte BlockIndex`
   Índice del primer bloque a salvar.
2. `byte BlockSize`
   Indica el tamaño del bloque (bytes) del transponder.
3. `byte BlocksNumber`
   Número de bloques a salvar.
4. `byte TagType`
   Según el transponder a usar tendrá un código u otro (por ejemplo 4 es para el modelo TFM05 1105.210).

9

```
void Putvalue(byte BlockNumber,
              byte BlockSize,
              byte Index,
              byte Value)
```

Descripción general

Este método establece el valor de una celda de memoria del tag a un determinado valor. Ha de tenerse en cuenta que esto solo modifica el valor del buffer interno de memoria que se ve entre operaciones de lectura y escritura, con lo que dicho valor no es salvado al transponder hasta que se efectúa una operación de `SaveTag`.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `byte BlockNumber`
   Indica el número de bloque del transponder.
2. `byte BlockSize`
   Indica el tamaño del bloque del transponder.
3. `byte Index`
   Índice dentro del bloque cuyo valor queremos modificar.
4. `byte Value`
   Valor que se desea asignar a la celda de memoria.

10

```
void Reset()
```

Descripción general

Este método permite reinicializar el funcionamiento del componente, así como reinicializar la comunicación con la antena y el modulo de bus. Debe ser llamado, por ejemplo, tras una operación con error.

Retorno

`void`

Este método tampoco retorna parámetros, sin embargo el sistema de control debería comprobar que `GetStatus` termina devolviendo un valor de 0 tras un tiempo.

Parámetros

Este método no requiere parámetros de entrada.

11

```
void SaveTag(byte BlockIndex,
             byte BlockSize,
             byte BlocksNumber,
             byte TagType)
```

Descripción general

Este método permite salvar los contenidos de un bloque de memoria al transponder que se encuentra en el rango de acción de la antena (siempre que este sea de tipo escritura).

Retorno

`void`

El método no retorna parámetros. El sistema de control deberá chequear el componente mediante el método `GetStatus` de forma periódica hasta obtener un código de operación completada.

Parámetros

1. `byte` `BlockIndex`
   Índice del primer bloque a salvar.
2. `byte` `BlockSize`
   Indica el tamaño del bloque (bytes) del transponder.
3. `byte` `BlocksNumber`
   Número de bloques a salvar.
4. `byte` `TagType`
   Según el transponder a usar tendrá un código u otro (por ejemplo 4 es para el modelo TFM05 1105.210).

12

```
void SetTimeOut(dword Timeout)
```

Descripción general

Este método establece el tiempo máximo, en milisegundos, que una operación puede estar realizándose sin respuesta antes de que el componente la aborte y señale un error de tiempo de espera agotado (código 8).

Retorno

`void`

El método no retorna ningún valor.

Parámetros

1. `dword` `Timeout`
   Tiempo en milisegundos de timeout para las operaciones de lectura, escritura y reset.

4.2.26INDITEX SE

4.2.26.1 Introducción

El propósito de este apartado es explicar el uso del componente *InditexSE*, que permite al programa de control comunicarse con el sistema de expedición de Tempe.

El sistema de expedición de Tempe incluye un protocolo de comunicaciones basado en sockets TCP para comunicarse con él. Mediante este protocolo, el sistema de control y el de expedición conocen sus estados, y se transfieren información como destinos de los transportes o estados de los mecanismos. En este sistema se identifican dos componentes principales:

El sistema de control (SC): encargado de ejecutar el sistema de control y de hacer las veces de maestro o servidor en las comunicaciones con el sistema de expedición.

- El sistema de expedición (SE): encargado de decidir los destinos de los transportes.

El componente *InditexSE* implementa la parte de control. En este protocolo se indica que el sistema de control tiene que mantener dos canales de comunicaciones, uno para cada sentido. A partir de este momento llamaremos canal SEND al canal que se utiliza para enviar información desde el CE al SE, y canal RECEIVE al que se utiliza para enviar información desde el SE al CE.

Es necesaria también la generación periódica de ciertos mensajes de control para el mantenimiento activo de la comunicación, así como acusar todas las tramas que se envían. Todo este trabajo lo realiza de manera interna el componente, siendo transparente desde el programa de control.

4.2.26.2 Configuración

Es necesario crear un fichero llamado TempeConfig.ini en el directorio de Galileo (c:\Galileo), en el que se indicarán todos los parámetros de configuración del componente. Este fichero se divide en cinco secciones que se detallan a continuación:

4.2.26.2.1 Sección "General"

En esta sección se indican parámetros generales de configuración del sistema de control y que tienen que ser los mismos que sus equivalentes en el sistema de expedición, ya que son parámetros que regulan el funcionamiento interno del protocolo:

- *RECEIVE_PORT*: puerto en el que escucha el servidor RECEIVE. Si no existe esta etiqueta, se utiliza por defecto el 6080.
- *SEND_PORT*: puerto en el que escucha el servidor SEND. Si no existe esta etiqueta, se utiliza por defecto el 6081.

- *KEEP_ALIVE_PERIOD*: perido de tiempo en milisegundos que se esperará para enviar un nuevo mensaje keep alive al SE, o en el espera recibir un mensaje keep alive desde el SE. Si no existe esta etiqueta se establece un valor por defecto de 8000 ms.
- *KEEP_ALIVE_EXPECTED*: periodo de tiempo en milisegundos que se esperará por un mensaje keep alive después de consumido el tiempo indicado por el parámetro anterior. Este parámetro permite dar un margen de tiempo antes de dar al SE como finalizado. Si no existe esta etiqueta se establece un valor por defecto de 4000 ms, por lo que finalmente, se tiene un periodo de espera por un keep alive desde el SE de 12000 ms.
- *RETRY_PERIOD*: periodo de tiempo en milisegundos en el que se espera recibir un mensaje de acuse (ACK) de una trama o se procederá a su reenvío. Si no existe esta etiqueta, se utiliza el valor por defecto de 2000 ms.
- *REINTENTOS*: número de veces que se reenvía una trama antes de dar por finalizado al SE. Si no existe esta etiqueta, se utiliza el valor de por defecto de dos intentos.
- *MAX_CLIENTS*: máximo número de clientes que permitimos que se conecten al sistema de control. Si no existe esta etiqueta, se utiliza el valor por defecto de 20 clientes.

4.2.26.2.2 Secciones "Sources" y "Destinations"

En estas secciones, de idéntica estructura, se especificarán los posibles orígenes y destinos de los transportes. Se trata de una lista de destinos (cadenas alfanuméricas de hasta 10 caracteres) y un índice numérico con el que será conocido dentro del programa de control.

4.2.26.2.3 Seccion "Mechs"

Esta sección es similar a las anteriores, pero referida a los nombres de los mecanismos. Se trata de una lista de mecanismos (cadenas alfanuméricas de hasta 10 caracteres) y un índice numérico con el que será conocido en el programa de control.

4.2.26.2.4 Seccion "Params"

En esta sección se indican los parámetros de configuración que el sistema de expedición podrá enviarnos, así como sus valores posibles. Consiste en una lista de nombres de parámetros y sus correspondientes valores separados por comas, de la forma:

```
Parámetro = valor1, valor2,…, valorn
```

El orden con el que se indiquen los parámetros y sus valores definirán el índice que con el serán conocidos en el programa de control. Así, el primer parámetro que se indique, será el cero, el siguiente el uno y así sucesivamente. Lo mismo ocurre con

los valores, teniendo en cuenta que la cuenta empieza con el primer valor del parámetro.

4.2.26.2.5 Ejemplo de archivo de configuración

```
[General]
RECEIVE_PORT=6080
SEND_PORT=6081
KEEP_ALIVE_PERIOD=8000
KEEP_ALIVE_EXPECTED=4000
RETRY_PERIOD=2000
REINTENTOS=2
MAX_CLIENTS=20

[Sources]
00010001=0
00010002=1
00010003=2

[Destinations]
00010001=0
00010002=1
00010003=2

[Mechs]
MECANISMO1=0
MECANISMO2=1

[Params]
blckchute=S
dumpchute=RECHAZO
```

4.2.26.3 Interface

Listado de métodos:

1	void AddContainerDate(word Handle, word Buffer, byte Data)
2	void AddContainerDestination(word Handle, word Buffer, word DestinationNumber)
3	void AddMechState(word Handle, word MechIndex, dword Data)
4	void AddScannerData(word Handle, word Scanner, word Data)
5	void DeleteContainerDate(word Handle, word Buffer)
6	void DeleteContainerDestinations(word Handle, word Buffer)
7	void DeleteMechState(word Handle, word MechIndex)
8	void DeleteScannerData(word Handle,

	word Scanner)
9	void DisposeContainerBuffer(word Handle, word Scanner)
10	word GetConnectionHandle(string Connection)
11	word GetContainerBuffer(word Handle, word Scanner)
12	word GetContainerDestination(word Handle, word Buffer, word Index)
13	word GetContainerDestinationsCount(word Handle, word Buffer)
14	word GetFreeContainerBuffer(word Handle)
15	word GetMechOrder(word Handle, byte &Type)
16	word GetParamValue(word Handle, word ParameterIndex)
17	word GetTransportOrder(word Handle, word Buffer)
18	bool hasMechOrder(word Handle)
19	bool hasTransportOrder(word Handle)
20	void Init()
21	bool IsReady()
22	void SendMechState(word Handle)
23	void SendTransportDimensions(word Handle, word Buffer, word Length, word High, word Width)
24	void SendTransportEnd(word Handle, word Buffer, word TargetIndex)
25	void SendTransportRequest(word Handle, word Buffer, word SourceIndex)
26	void SendTransportWeigth(word Handle, word Buffer, word Weigth)
27	void SetHardReset(word Handle, string PLCName)
28	void SetMechState(word Handle, word State)

Descripción de los métodos:

1

```
void AddContainerDate(word Handle,
                      word Buffer,
                      byte Data)
```

Descripción general

Añade un `byte` al área de datos del contenedor

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word` `Handle`
   Manejador de una conexión.
2. `word` `Buffer`
   Área de datos reservada para ese contenedor.
3. `byte` `Data`
   Byte que se desea añadir.

2

```
void AddContainerDestination(word Handle,
                             word Buffer,
                             word DestinationNumber)
```

Descripción general

Añade un destino al área de datos de contenedor.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word` `Handle`
   Manejador de una conexión.
2. `word` `Buffer`
   Área de datos reservada para ese contenedor.
3. `word` `DestinationNumber`
   Número de destino.

3

```
void AddMechState(word Handle,
                  word MechIndex,
                  dword Data)
```

Descripción general

Añade un `dword` a los datos del mecanismo.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word` `Handle`
   Manejador de una conexión.
2. `word` `MechIndex`
   Índice del mecanismo.
3. `dword` `Data`
   Datos que se desean añadir.

4

```
void AddScannerData(word Handle,
                    word Scanner,
                    word Data)
```

Descripción general

Añade un `byte` al área de datos de escáner.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word` Handle

   Manejador de una conexión.
2. `word` Scanner

   Área de datos reservada para el escáner indicado.
3. `word` Data

   Byte de datos que se desea añadir.

5

```
void DeleteContainerDate(word Handle,
                         word Buffer)
```

Descripción general

Borra el área de datos del contenedor.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word` Handle

   Manejador de una conexión.
2. `word` Buffer

   Área de datos reservada para ese contenedor.

6

```
void DeleteContainerDestinations(word Handle,
                                 word Buffer)
```

Descripción general

Borra todos los destinos del área de datos de contenedor.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word` Handle

   Manejador de una conexión.
2. `word` Buffer

Área de datos reservada para ese contenedor.

7

```
void DeleteMechState(word Handle,
                     word MechIndex)
```

Descripción general

Borra el área de datos de un mecanismo.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word` `Handle`

   Manejador de una conexión.

2. `word` `MechIndex`

   Índice del mecanismo.

8

```
void DeleteScannerData(word Handle,
                       word Scanner)
```

Descripción general

Borra los datos de un área de datos del escáner indicado.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word` `Handle`

   Manejador de una conexión.

2. `word` `Scanner`

   Área de datos reservada para el escáner indicado.

9

```
void DisposeContainerBuffer(word Handle,
                            word Scanner)
```

Descripción general

Libera un área de datos de contenedor.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word` `Handle`

   Manejador de una conexión.

2. `word` `Scanner`

   El manejador del área de datos que se quiere liberar.

10

```
word GetConnectionHandle(string Connection)
```

Descripción general

Retorna un `word` con el manejador de la conexión que se pasa como parámetro.

Retorno

`word`

Retorna un manejador mayor o igual que cero en caso de existir conexión. Si no existe la conexión retornaría el valor `-1`

Parámetros

1. `string` Connection
   Nombre del cliente con el que se desea comunicar.

11

```
word GetContainerBuffer(word Handle,
                        word Scanner)
```

Descripción general

Retorna un `word` con el manejador de una área de datos de contenedor, que contenga los datos del contenedor que se encuentra en el área de scanner especificada.

Retorno

`word`

Retorna un manejador mayor o igual que cero en caso de existir conexión. Si no existe la conexión retornaría el valor `-1.`

Parámetros

1. `word` Handle
   Manejador de una conexión.
2. `word` Scanner
   El manejador del área del escáner, típicamente el número de máquina.

12

```
word GetContainerDestination(word Handle,
                             word Buffer,
                             word Index)
```

Descripción general

Devuelve el destino almacenado en la posición index del área de datos.

Retorno

`word`

Devuelve el destino seleccionado.

Parámetros

1. `word` Handle
   Manejador de una conexión.
2. `word` Buffer
   Área de datos reservada para ese contenedor.
3. `word` Index
   Posición en la que se almacena el destino.

13

```
word GetContainerDestinationsCount(word Handle,
                                   word Buffer)
```

Descripción general

Devuelve el número de destinos almacenados en el área de datos.

Retorno

`word`

Devuelve el número de destinos.

Parámetros

1. `word` `Handle`
   Manejador de una conexión.
2. `word` `Buffer`
   Área de datos reservada para ese contenedor.

14

```
word GetFreeContainerBuffer(word Handle)
```

Descripción general

Retorna un `word` con el manejador de un área de datos de contenedor. En caso de no haber ningún área libre, sobrescribe la más antigua

Retorno

`word`

Retorna un manejador mayor o igual que cero en caso de existir conexión.

Parámetros

1. `word` `Handle`
   Manejador de una conexión.

15

```
word GetMechOrder(word Handle,
                  byte &Type)
```

Descripción general

Recibe una orden de limpieza.

Retorno

`word`

Retorna el índice del mecanismo que recibe la orden.

Parámetros

1. `word` `Handle`
   Manejador de una conexión.
2. `byte` `Type`
   Tipo de orden que se ha de realizar pasado por referencia:
   a. `0` → Arranque
   b. `1` → Parada
   c. `2` → Reset

d. 3 → Clear

16

```
word GetParamValue(word Handle,
                   word ParameterIndex)
```

Descripción general

Devuelve el valor asignado a un parámetro.

Retorno

word

Retorna el índice del valor del parámetro.

Parámetros

1. word Handle
   Manejador de una conexión.
2. word ParameterIndex
   Índice el parámetro que se desea consultar.

17

```
word GetTransportOrder(word Handle,
                       word Buffer)
```

Descripción general

Recibe una orden de transporte para el contenedor que indica el área de datos.

Retorno

word

Retorna el índice del contenedor.

Parámetros

1. word Handle
   Manejador de una conexión.
2. word Buffer
   Área de datos reservada para el contenedor.

18

```
bool hasMechOrder(word Handle)
```

Descripción general

Comprueba si existen ordenes de mecanismos a tratar.

Retorno

true

En caso de que existan órdenes.

false

En caso de que NO existan órdenes.

Parámetros

1. word Handle
   Manejador de una conexión.

19

```
bool hasTransportOrder(word Handle)
```

Descripción general

Comprueba si hay ordenes de transporte pendientes.

Retorno

`true`
Existen órdenes de transporte.
`false`
NO existen órdenes de transporte.

Parámetros

1. `word` Handle
   Manejador de una conexión.

20

```
void Init()
```

Descripción general

Inicializa el sistema global.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

21

```
bool IsReady()
```

Descripción general

Comprueba si el sistema, en concreto los servidores, están listos y preparados para comenzar a trabajar.

Retorno

`true`
Los servidores están correctamente instalados y funcionando.
`false`
Los servidores NO están correctamente instalados.

Parámetros

Este método no requiere parámetros de entrada.

22

```
void SendMechState(word Handle)
```

Descripción general

Envia los datos del mecanismo.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word` Handle
   Manejador de una conexión.

23

```
void SendTransportDimensions(word Handle,
                             word Buffer,
                             word Length,
                             word High,
                             word Width)
```

Descripción general

Envia las dimensiones del contenedor que indica el área de datos.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `word` Handle
   Manejador de una conexión.
2. `word` Buffer
   Área de datos reservada para ese contenedor.
3. `word` Length
   Longitud del contenedor.
4. `word` High
   Altura del contenedor.
5. `word` Width
   Ancho del contenedor.

24

```
void SendTransportEnd(word Handle,
                      word Buffer,
                      word TargetIndex)
```

Descripción general

Envia un fin de orden para el contenedor que indica el área de datos.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `word` Handle
   Manejador de una conexión.
2. `word` Buffer
   Área de datos reservada para ese contenedor.
3. `word` TargetIndex
   Índice del destino al que llegó el contenedor.

25

```
void SendTransportRequest(word Handle,
                          word Buffer,
                          word SourceIndex)
```

Descripción general

Envia una petición de orden para un contenedor.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word` `Handle`

   Manejador de una conexión.
2. `word` `Buffer`

   Área de datos reservada para este contenedor.
3. `word` `SourceIndex`

   Índice del origen en el que está el contenedor.

26

```
void SendTransportWeigth(word Handle,
                         word Buffer,
                         word Weigth)
```

Descripción general

Envia el peso del contenedor que indica el área de datos.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word` `Handle`

   Manejador de una conexión.
2. `word` `Buffer`

   Área de datos reservada para ese contenedor.
3. `word` `Weigth`

   Peso del contenedor.

27

```
void SetHardReset(word Handle,
                  string PLCName)
```

Descripción general

Envía una notificación de *hard reset*.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word` `Handle`

   Manejador de una conexión.
2. `string` `PLCName`

Nombre del PLC que ha sufrido el HardReset.

28

```
void SetMechState(word Handle,
                  word State)
```

Descripción general

Establece el estado de un mecanismo.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word` `Handle`
   Manejador de una conexión.
2. `word` `State`
   Estado del mecanismo:
   a. `0` → Arrancado
   b. `1` → Parado
   c. `-1` → Error

4.2.26.4 Áreas de datos accesibles

El componente dispone de dos áreas de datos que permiten al usuario trabajar de manera sencilla con los contenedores y mecanismos. Para cada cliente que se conecte con el sistema de control, se mantienen dos zonas de datos, una para los mecanismos y otra para los contenedores. La zona de datos de contenedores está pensada para que el usuario escriba en ella el identificador del contenedor y los posibles destinos de éste, de manera que los métodos antes mostrados trabajarían siempre con esta área. Los métodos que utilizan esta área son:

- *GetFreeContainerBuffer*: permite reservar un área de datos de contenedor
- *DisposeContainerBuffer*: libera un área de datos reservada
- *AddContainerDestination*: Añade un destino a un área de datos
- *DeleteContainerDestinations*: Elimina los destinos del área de datos
- *GetContainerDestinationsCount*: devuelve el número de destino actualmente almacenados en el área de datos
- *GetContainerDestination*: devuelve el destino especificado
- *AddContainerData*: permite añadir `bytes` al área
- *DeleteContainerData*: permite eliminar el área
- *SendTransportReques*: envía una petición de transporte para el contenedor que esté cargado en el área de datos

- *SendTransportEnd*: envía la orden de fin de transporte para el contenedor cargado en el área de datos.
- *SendTransportWeigth*: envía el peso del transporte cargado en el área de datos
- *SendTransportDimensions*: envía las dimensiones del transporte cargado en el área de datos
- *GetTransportOrder*: recibe una orden de transporte para el contenedor en curso

Como complemento de esta área, existe un número fijo de *áreas de escáner*, pensadas únicamente como almacén temporal de los datos procedentes de las lecturas de los escáneres. Estas áreas no requieren ser reservadas por parte del usuario, ya que los escáneres son fijos en la instalación de manera, que a priori, se les asignaría a cada scanner un índice con el que serán referenciados en el programa de control. El objetivo de estás áreas de scanner es permitir la búsqueda de los datos de un contenedor sin tener que consultar al sistema de expedición, de manera que se evitan realizar una comunicación de red. Las funciones que trabajan con estas áreas son las siguientes:

- *AddScannerData*: añade un carácter a un área de escáner
- *DeleteScannerData*: borra el área de escáner seleccionado
- *GetContainerBuffer*: utilizando como referencia el área de scanner que se le pasa como parámetro, busca un área de datos de contenedor que coincida.

Estas dos zonas son persistentes, de manera que ante posibles fallos eléctricos, los datos no se pierden. Además, son gestionadas de manera automática por el propio componente, de manera que en caso de no liberarse de la manera adecuada las áreas de datos de contenedor, estas son reutilizadas en función de su antigüedad.

La zona de datos de mecanismos esta pensada como una colección de `dword` en las que el usuario irá cargando las informaciones que crea necesarias y que tendrán que ser enviadas periódicamente al SE mediante una trama SSIF (consultar especificación de las comunicaciones). Para el control de éste área, se tienen los siguientes métodos:

- *AddMechData*: añade un nuevo `dword` a la colección.
- *DeleteMechData*: elimina todos los `dword` de la colección
- *SendMechData*: Genera la trama SSIF con los datos del área y la envía

4.2.26.5 Utilización del componente

4.2.26.5.1 Inicialización

El componente necesita ser inicializado de manera explicita mediante la invocación del método `Init`. Esto tiene que realizarse al menos una vez, por lo que lo ideal es hacerlo en las rutinas de arranque frío y caliente del secuenciador.

Una vez invocado este método, el componente comenzará la inicialización. Puesto que internamente prepara varios servidores, la inicialización no es inmediata. Es importante no realizar operaciones con el componente mientras no esté correctamente iniciado. Para ello, se dispone del método `isReady`, que permite saber cuando está listo para trabajar.

Como el componente permite múltiples conexiones (por ejemplo, SE y Scanner Volumétrico), es necesario un sistema que permita el identificarlas. Se utilizan para ello handles que son asociados a las conexiones. Es necesario por tanto, antes de llamar a un método, el obtener un handle válido. Se utiliza para ello el método `GetConnectionHandle`. Se puede utilizar el siguiente código como estándar para el uso de los métodos del componente:

```
if (!this.p_Tempe->isReady()) return;
handle=this.p_Tempe->GetConnectionHandle(this.p_NombreSE);
if (handle<0) return;
```

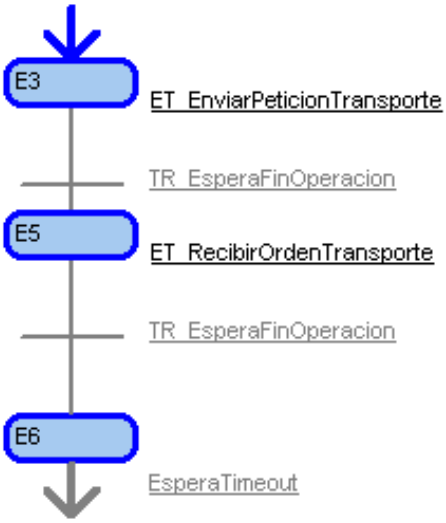
Donde p_Tempe es precisamente un parámetro del secuenciador de tipo InditexSE, y p_NombreSE es la cadena que identifica al sistema de expedición.

4.2.26.5.2 Gestión de transportes

4.2.26.5.2.1 Caso general

La gestión de los transportes se divide en tres partes: envíos de peticiones de órdenes, gestión de órdenes y finalización de las órdenes. Como se dijo antes, es necesario tener un handle a la conexión activa, para lo que se puede utilizar el código antes mostrado.

La petición de ordenes es el primer paso, y consiste en enviar al SE una trama en la que se indica un contenedor y un origen, esperando a que éste nos conteste con uno o varios destinos.



En la imagen anterior, se puede ver el grafo que permite tratar órdenes. El código de las funciones podría ser similar a este:

ET_ENVIARPETICIONTRANSPORTE

```
dword handle;
word i;
byte car;

this.p_OperacionRealizada=false;

if (!this.ServidoresReady()) return;

handle=this.p_Tempe->GetConnectionHandle(this.p_NombreSE);

if (handle<0)
   return;

if (this.p_Buffer<0) {// No tenemos buffer, lo pedimos
   this.p_Buffer=this.p_Tempe->GetFreeContainerBuffer(handle);
   if (this.p_Buffer<0) return;
}

this.p_Tempe->DeleteContainerData(handle,this.p_Buffer);

car=48; // El caracter '0'
for (i=0;i<19;i++) {
    this.p_Tempe->AddContainerData(handle,this.p_Buffer,car);
}
this.p_Tempe->AddContainerData(handle,this.p_Buffer,car+1);

// Enviamos el peso y las dimensiones del contenededo
this.p_Tempe->SendTransportWeigth(handle,this.p_Buffer,30);
this.p_Tempe->SendTransportDimensions(handle,this.p_Buffer,25,60,50);

// Enviamos la petición de transporte
```

```
this.p_Tempe->SendTransportRequest(handle,this.p_Buffer,1);

this.p_OperacionRealizada=true;
```

La primera parte del código simplemente sirve para obtener un handle válido, como se vio antes. Antes de poder pedir la orden es necesario disponer de un área de datos de contenedor libre en la que almacenemos los destinos. Para ello se realiza la llamada a *GetFreeContainerBuffer*. Después se carga en el área de datos del contenedor el identificador del contenedor para el que se pide orden, en este caso el "0000000000000000001". Después, a modo de ejemplo, se envían las dimensiones y el peso del contenedor. Por último, se envía la orden.

ET_RECIBIR_ORDEN_TRANSPORTE

```
word destino;
dword handle;

this.p_OperacionRealizada=false;

if (!this.HayOrdenTransporte()) return;

handle=this.p_Tempe->GetConnectionHandle(this.p_NombreSE);

if (handle<0)
   return;

// Almacenamos los destinos en el area de datos

destino=this.p_Tempe->GetTransportOrder(handle,this.p_Buffer);
while (destino>=0) {
      this.p_Tempe-
>AddContainerDestination(handle,this.p_Buffer,destino);
      destino=this.p_Tempe->GetTransportOrder(handle,this.p_Buffer);
}

// Enviamos el fin de orden para el primer destino recibido
destino=this.p_Tempe->GetContainerDestination(handle,this.p_Buffer,0);
this.p_Tempe->SendTransportEnd(handle,this.p_Buffer,destino);
// Liberamos el area de datos
this.p_Tempe->DisposeContainerBuffer(handle,this.p_Buffer);
this.p_Buffer=-1;

this.p_OperacionRealizada=true;
```

Para la recepción, comprobamos si existen órdenes y en caso de haberlas, llamamos a `GetTransportOrder` para obtener un destino. Es importante tener en cuenta que el método `hasTransportOrden` (que se llama desde HayOrdenTransporte) devuelve `true` en caso de haber órdenes, aunque sean para otro contenedor. Por tanto será necesario comporbar que el destino devuelto no es -1, que significará que no existen ordenes para ese contenedor. La función `GetTransportOrden` nos devuelve el primero destino recibido para ese contenedor.

El SE pudo enviar varios destinos (principal y lternativos) de manera que se necesitan varias llamadas a este método para obtenerlas todas. Por último, y a modo de ejemplo, se envía un fin de orden indicando que el transporte se ha llevado al primer destino que nos indicó el sistema de expedición. Para ello, simplemente obtenemos el destino mediante *GetContainerDestination* y enviamos el fin de orden mediante *SendTransportEnd*.

La función TR_Espera_Fin_Operación simplemente comprueba que el parámetro p_OperaciónRealizada esta a `true`, lo que nos permitirá cambiar de etapa. Además, en el ejemplo anterior se simuló una determinada carga de trabajo mediante el uso de temporizadores, y está función se encarga también de controlar estas temporizaciones.

4.2.26.5.2.2 Gestión de transportes sin consultar el sistema de expedición

Existen casos en los que no será necesario realizar consultas al sistema de expedición por que los destinos ya los tenemos almacenados. Estos casos tienen un desarrollo similar al anterior, pero con ciertas peculiaridades que se comentan a continuación.

1. Al igual que en el caso general, cuando nos llega un nuevo contenedor para el que no tengamos datos, tendremos que realizar un `GetFreeContainerBuffer`, de manera que obtengamos un área de contenedores libre.
2. En el supuesto que en esa máquina no se realice el fin de orden, no se tendrá que liberar el área de datos de contenedor.
3. Cuando ese contenedor llegue a otra máquina, será necesario recuperar el área de datos de contenedor basándose en la lectura del scanner. Para ello, será necesario el siguiente código:

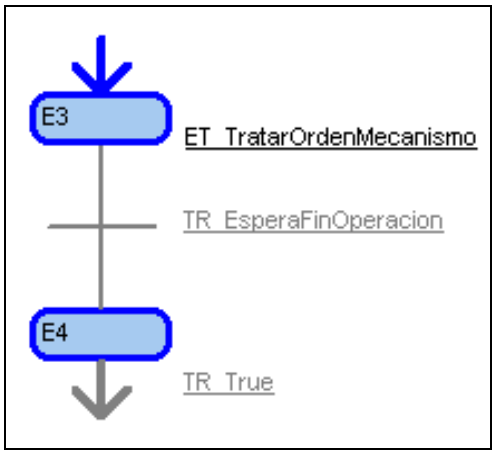
```
dword handle;
word i;
byte car;

if (!this.ServidoresReady()) return;

handle=this.p_Tempe->GetConnectionHandle(this.p_NombreSE);

if (handle<0)
   return;

// Cargamos los datos de la lectura del scanner en un area de scanner
this.p_Tempe->DeleteScannerData(handle,this->Number());

car=48; // El caracter '0'
for (i=0;i<19;i++) {
    this.p_Tempe->AddScannerData(handle,this->Number(),car);
}
this.p_Tempe->AddContainerData(handle,this->Number(),car+1);

// Obtenemos el area de datos de ese contenedor
```

```
this.p_Buffer=this.p_Tempe->GetContainerBuffer(handle,this->Number());
if (this.p_Buffer<0) return;
```

4. Al igual que en el caso general, cuando se genere el fin de orden es necesario liberar el área de datos de contenedor.

4.2.26.5.3 Gestión de los mecanismos

La gestión de los mecanismos se realiza de manera similar a la de los transporte. La diferencia básica es que no necesita que el sistema de control pida ordenes, sino que es el sistema de expedición la que las manda cuando son necesarias.



ET_TRATAR_ORDEN_MECANISMO

```
dword handle;
byte tipoOp;
dword mecanismo;

this.p_OperacionRealizada=false;

if (!this.ServidoresReady()) return;
handle=this.p_Tempe->GetConnectionHandle(this.p_NombreSE);
if (handle<0)  return;

if (this.HayOrdenesMecanismo()) {
   mecanismo=this.p_Tempe->GetMechOrder(handle,tipoOp);

   // Actualizamos el estado del mecanismo
   if (tipoOp==0) // Orden de arranque
      this.p_Tempe->SetMechState(mecanismo,0);
   if (tipoOp==1) // Orden de parada
      this.p_Tempe->SetMechState(mecanismo,1);

   if ((tipoOp==2)||(tipoOp==3)) {
      // Es una orden de reset o una de limpieza de mecanismo
   }
}
```

```
this.p_OperacionRealizada=true;
```

Como antes, la primera parte simplemente obtiene un handle válido. Después se comprueba si existen órdenes de mecanismo. Si las hay, a modo de ejemplo, en el caso de ser una orden de arranque o de parada se llama a la función `SetMechState` para cargar el nuevo estado.

HAY_ORDEN_MECANISMO

```
dword handle;
handle=this.p_Tempe->GetConnectionHandle(this.p_NombreSE);
if (handle<0) return false;
return (this.p_Tempe->hasMechOrder(handle));
```

4.2.26.5.4 Tratamiento de los parámetros enviados por el sistema de expedición

El sistema de expedición puede enviar en cualquier momento un mensaje informando los nuevos valores de los parámetros. El Sistema de Control Galileo gestiona automáticamente estos mensajes, almacenando el nuevo valor del parámetro, de manera que esté disponible para el usuario. De esta manera, la consulta de un parámetro desde el sistema de control se puede hacer en cualquier momento, sin que suponga una transmisión ni un consumo de tiempo. Para ello se utiliza la función *`GetParamValue`*, a la que se le pasa el índice de la conexión en la que se realizará la consulta y el índice del parámetro que se consulta. La función retornará el índice de valor que tiene ese parámetro.

4.2.26.5.5 Tareas automatizadas

El componente InditexSE, realiza ciertas tareas de manera automática, de manera que son invisibles al usuario. La mayoría de estas tareas tienen que ver con el mantenimiento activo de la comunicación, y son:

- Acuse de las tramas enviadas desde el cliente
- Reenvió en caso de error
- Envío de tramas KeepAlive

Existen además ciertas tareas relacionadas con el control de los mecanismos. En concreto, la función `SetMechState`, que carga el estado de un mecanismo, es necesario que se invoque cada vez que el usuario arranque o pare un máquina, ya que el SE puede consultar este estado, y la respuesta se hará de manera automática sin intervención del usuario en función de los datos cargados con esta función. Otra tarea que se realiza de manera automática es la gestión de los parámetros que el se envían al sistema de control, limitando la intervención del usuario a consultarlos cuando los necesite.

4.2.27 METTLER TOLEDO: ESCÁNER VOLUMÉTRICO

4.2.27.1 Interface

Para poder realizar control de peso y dimensiones en ciertas instalaciones, es necesario incluir un sistema de bascula que nos de una lectura sobre el peso y las medidas de una unidad de almacenaje. Para este cometido se puede utilizar el scanner volumétrico diseñado por Mettler Toledo.

ClasS	VolumetricScannerMettler
BUS IN	0 bytes
BUS OUT	0 bytes

Listado de métodos:

1	`bool ConfigScanner(dword Port, word Timeout, string Host)`
2	`byte GetData(word &Weight, word &Length, word &High, word &Width)`
3	`dword GetStatus()`
4	`void Reset()`

Descripción de los métodos:

1

```
bool ConfigScanner(dword Port,
                   word Timeout,
                   string Host)
```

Descripción general

Configura el componente con los datos necesario para realizar la conexión con el escáner.

Retorno

`true`
Es posible configurar el escáner con los parámetros indicados.
`false`
NO es posible configurar el escáner con los parámetros indicados.

Parámetros

1. `dword Port`
   Puerto TCP en el que está escuchando el escáner. Por defecto es el 2000.
2. `word Timeout`
   Tiempo en milisegundos tras los que sin recibir ninguna trama por parte del escáner, se da por perdida la conexión y se trata de volver a conectar.
3. `string Host`

Dirección IP del escáner.

2

```
byte GetData(word &Weight,
             word &Length,
             word &High,
             word &Width)
```

Descripción general

Obtiene los resultados de la lectura del escáner volumétrico.

Retorno

`byte`

Se indica el estado del escáner y la validez de la lectura obtenida:

a. `0` → Escáner online y lectura válida
b. `1` → Escáner online
c. `2` → Escáner online con lectura inválida. En este caso, los motivos de la lectura inválida pueden ser bien que el escáner no haya medido correctamente o bien que no exista ningún objeto que medir.

Parámetros

1. `word Weight`
   Parámetro de salida en el que se almacenará el peso en gramos.
2. `word Length`
   Parámetro de salida que almacenará el largo del elemento en milímetros.
3. `word High`
   Parámetro de salida que almacenará el alto del elemento en milímetros.
4. `word Width`
   Parámetro de salida que almacenará el ancho del elemento en milímetros.

3

```
dword GetStatus()
```

Descripción general

Evalúa el estado del escáner.

Retorno

`dword`

Retorna el estado del escáner.

Parámetros

Este método no requiere parámetros de entrada.

4

```
void Reset()
```

Descripción general

Resetea el valor almacenado como última lectura. No es necesario invocarlo tras una lectura, salvo que se tengan problemas por leer varias veces la misma lectura.

Retorno

`void`

Este método no retorna ningún valor.
Parámetros
Este método no requiere parámetros de entrada.

4.2.28TIMER

4.2.28.1 Introducción

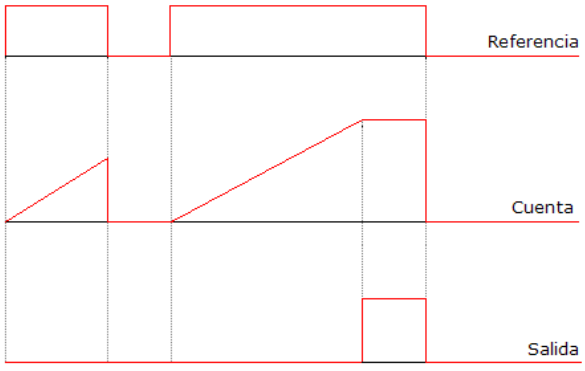
NOTA IMPORTANTE: Este componente está "Deprecado", es decir, marcado como obsoleto, por lo que su uso no se recomienda, dado que en el futuro podría dejar de ser incluido en la lista de componentes entregados con el sistema Galileo. Ello implica que recibiremos avisos del compilador informando de este hecho (no quiere decir que no funcione, simplemente avisa que en futuro puede no soportarse ese componente, asi que es mejor usar otra opción a fin de tener más compatibilidad). Concretamente, se deberian usar los Timers que implementa Machine en lugar de este componente.

Este objeto permite definir un temporizador y tiene varios modos de funcionamiento.

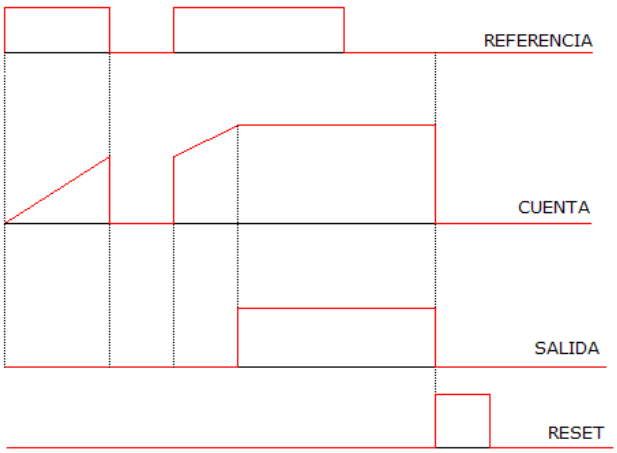
Debe hacerse notar que la precisión del Timer es de 1 milisegundo, ya que es la máxima precisión que en Windows NT se soporta sin realizar operaciones a bajo nivel.

Existen varias posibilidades de configuración del Timer. Estas son:

- *No memorizado - TIPO 0*



- *Memorizado - TIPO 1*



4.2.28.2 Interface

Este componente expone el siguiente interface

Class	Timer
BUS IN	0 bytes
BUS OUT	0 bytes

Listado de métodos:

1	void ChangeResolution(word Resolution, word Type)
2	bool Elapsed()
3	void Restart()
4	void SetInvTimer(bool &Signal, dword Value, word Type, word Resolution)
5	void SetTimer(bool &Signal, dword Value, word Type, word Resolution)
6	dword Value()

Descripción de los métodos:

1

```
void ChangeResolution(word Resolution,
                      word Type)
```

Descripción general

Este método permite el cambio de resolución y tipo de temporizador sobre un temporizador previamente configurado.

Retorno

`void`

Este método no devuelve ningún valor.

Parámetros

1. `word Resolution`
   Resolución a la que se desea cambiar.
2. `word Type`
   Nuevo tipo de temporizador que se desea asignar.

2

```
bool Elapsed()
```

Descripción general

Método que se utilizará para averiguar si la cuenta del temporizador ha expirado.

Retorno

`true`

La cuenta ha alcanzado el valor configurado.

`false`

La cuenta aún no ha finalizado.

Parámetros

Este método no requiere parámetros de entrada.

3

```
void Restart()
```

Descripción general

Este método fuerza el reinicio de la cuenta del temporizador.

Retorno

`void`

Este método no devuelve ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

4

```
void SetInvTimer(bool &Signal,
                 dword Value,
                 word Type,
                 word Resolution)
```

Descripción general

Ídem al anterior, la única diferencia está en que se tomará la señal de referencia con lógica negativa.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `bool Signal`
   Variable booleana por referencia que genera el disparo. NOTA: Esta variable debe ser global pasar una variable local por referencia sólo traería como consecuencia una caída del sistema.
2. `dword Value`
   Valor de cuenta del timer, se refiere en todo caso a las unidades de cuenta establecidas en la resolución.
3. `word Type`
   Tipo de temporizador (0 no memorizado y 1 memorizado).
4. `word Resolution`
   Resolución en milisegundos, indica como se va a realizar la cuenta.

5

```
void SetTimer(bool &Signal,
              dword Value,
              word Type,
              word Resolution)
```

Descripción general

Este método debe invocarse en las rutinas de arranque, en el se establece la señal de disparo del temporizador, el valor del temporizador, el tipo de temporizador y la resolución.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `bool Signal`
   Variable booleana por referencia que genera el disparo. NOTA: Esta variable debe ser global pasar una variable local por referencia sólo traería como consecuencia una caída del sistema.
2. `dword Value`
   Valor de cuenta del timer, se refiere en todo caso a las unidades de cuenta establecidas en la resolución.
3. `word Type`
   Tipo de temporizador (0 no memorizado y 1 memorizado).
4. `word Resolution`
   Resolución en milisegundos, indica como se va a realizar la cuenta.

6

```
dword Value()
```

Descripción general

Método que se utilizará para averiguar el valor actual de cuenta del temporizador.

Retorno

dword
Devuelve la cuenta actual de pulsos del temporizador.
Parámetros
Este método no requiere parámetros de entrada.

4.2.29 LJU: RAILBUS SYSTEM Y DKZCOMP

4.2.29.1 Introducción

Los componentes `RailBus` y `DKZComp` permiten controlar el funcionamiento de una electrovía basada en los dispositivos desarrollados por LJU. Debido a ciertas características de estos dispositivos, los componentes necesarios para su utilización en el Sistema de Control Galileo tienen ciertas diferencias en cuanto a su manejo respecto a otros componentes. Es necesario por tanto, explicar las características básicas de estos dispositivos para así comprender el manejo de los componentes.

4.2.29.2 Componentes de la electrovía

La electrovía se componente de tres elementos básicos:

- *DKZ*: elemento "inteligente" y que mantiene una comunicación directa con el sistema de control. Además de este enlace con el sistema de control (mediante Profibus), mantiene los siguiente enlaces:
  - Enlace con los carros de la instalación mediante el bus electrificado.
  - Enlace con los controladores de desvíos mediante un bus 485.
  - Enlace con otros DKZ mediante los módulos IKB.
- *Controlador de desvío*: dispositivo que controla la posición de los desvíos de la instalación.
- *Controlador de carro*: dispositivo que se monta en los carros y los controla durante su movimiento.

Como se puede ver, el único elemento que se comunica con el sistema de control, en este caso Galileo, es el DKZ. Esto implica que todas las operaciones, sean referentes a carros, desvíos o DKZ tienen que realizarse interactuando con éste. Además, en una instalación se pueden disponer de varios DKZ, que se comunican entre sí, y que intercambian información entre ellos de manera transparente.

La posibilidad de que varios DKZ controlen una electrovía añade un problema a la hora de controlarla desde Galileo. La manera natural sería utilizar un único componente, de manera que fuese transparente el número de DKZ que existan. Sin embargo, al ser éstos los únicos que tienen conexión con el sistema de control hace necesario que sean instanciados en el bus (cada uno tiene su dirección Profibus y su espacio de E/S). De ahí que existan dos componentes para la gestión de la electrovía:

- *DKZComp*: componente que controla un DKZ. SOLO ES NECESARIO POR MOTIVOS FORMALES Y PARA LA INICIALIZACIÓN. Durante el resto de la ejecución no se utilizará.
- *RailBus*: componente que realmente se utiliza para controlar la electrovía.

4.2.29.3 Interface

El componente DKZComp exporta la siguiente interface

Class	DKZComp
BUS IN	228 bytes
BUS OUT	130 bytes

Listado de métodos:

```
1  dword GetMyId()
```

1

```
dword GetMyId()
```

Descripción general

Devuelve el identificado de ese DKZ en el sistema.

Retorno

dword
Retorna el identificador del DKZ en el sistema.

Parámetros

Este método no requiere parámetros de entrada.

El componente *RailBus* exporta la siguiente interface

Class	RailBus
BUS IN	0 bytes
BUS OUT	0 bytes

Listado de métodos del componente:

```
1  void AddDKZ(dword Id,
              byte DKZAddress)
2  void CarriageAutomatic(word TrolleyAddress)
3  void CarriageBackwardFastManual(word TrolleyAddress)
4  void CarriageBackwardSlowManual(word TrolleyAddress)
5  void CarriageForwardFastManual(word TrolleyAddress)
```

6	void CarriageForwardSlowManual(word TrolleyAddress)
7	bool ConvChainEmpty(word TrolleyAddress) bool ConvChainFull(word TrolleyAddress) bool ConvChainRunning(word TrolleyAddress) bool ConvChainStopped(word TrolleyAddress)
8	void ConveyorLeftManual(word TrolleyAddress)
9	void ConveyorLoadLeft(word TrolleyAddress)
10	void ConveyorLoadRight(word TrolleyAddress)
11	void ConveyorRightManual(word TrolleyAddress)
12	void ConveyorUnLoadleft(word TrolleyAddress)
13	void ConveyorUnLoadRight(word TrolleyAddress)
14	void DisableTrolleyTA(word TrolleyAddress)
15	void DKZAutomatic(byte DKZAddress)
16	void DKZManual(byte DKZAddress)
17	void DKZReset(byte DKZAddress)
18	void EnableTrolleyTA(word TrolleyAddress)
19	bool ExistsTrolley(word TrolleyAddress)
20	bool Failure(word TrolleyAddress)
21	bool GetBufferState(word Buffer)
22	byte GetConveyorChainStatusCode(word TrolleyAddress)
23	word GetDestinationSegmentFrom(word SegmentNumber, word Delta)
24	word GetDistanceBetweenSegments(word StartSegment, word EndSegment)
25	word GetDistanceBetweenSegments2(word StartSegment, dword RealStartPosition, word EndSegment, dword RealEndPosition)
26	byte GetDKZCounter()
27	word GetMachineSegment(word MachineNumber)
28	word GetNearestStationToTrolley(word TrolleyAddress)
29	word GetNearestTrolley(word SegmentNumber)
30	word GetNearestTrolleyToStation(byte StationNumber, byte StationType, byte Substation)
31	word GetNearestTrolleyToStationByMachine(dword NearestStation bool Search)
32	word GetNearestTrolleyToStationEx(byte StationNumber, byte StationType, byte Substation, byte MinDistance)
33	word GetNearestTrolleyToTrolley(byte TrolleyAddress)
34	word GetNextSegmentWithOffset(word StartSegment, word OffsetPosition)
35	word GetSegment(word Position)
36	bool GetSegmentData(word SegmentNumber, word &InitPosition, dword &EndPosition)
37	word GetSubStationCount(byte StationNumber, byte StationType)
38	word GetTrolleyCapacity(word TrolleyAddress)
39	byte GetTrolleyControlA(word TrolleyAddress)

40	byte GetTrolleyControlB(word TrolleyAddress)
41	word GetTrolleyCurrentOccupation(word TrolleyAddress)
42	word GetTrolleyCurrentSpeed(word TrolleyAddress)
43	void GetTrolleyDestinationBuffer(word TrolleyAddress word &SegmentBuffer, dword &Position)
44	dword GetTrolleyDistanceToPos(word TrolleyAddress, word Segment, dword Position)
45	byte GetTrolleyDKZ(word TrolleyAddress)
46	byte GetTrolleyErrorNumber(word TrolleyAddress)
47	byte GetTrolleyFlags(word TrolleyAddress)
48	word GetTrolleyInTroubleWith(word TrolleyAddress)
49	dword GetTrolleyPosition(word TrolleyAddress)
50	word GetTrolleyRealSegment(word TrolleyAddress)
51	word GetTrolleyRoutePoint(word TrolleyAddress)
52	byte GetTrolleyStatusCode(word TrolleyAddress)
53	word GetTrolleyTroubleDistance(word TrolleyAddress)
54	void Init()
55	bool InPosition(word TrolleyAddress)
56	void InvalidateMovements(word TrolleyAddress)
57	bool IsAllowedLoad(dword UnloadMachine, dword LoadMachine)
58	bool IsASwitch(word aSegment, dword &SwitchAddress, dword &EntancesCount, dword &ExitsCount)
59	bool isTrolley_NOCOMM(word TrolleyAddress) bool isTrolley_MANUALMODE(word TrolleyAddress) bool isTrolley_ERROR(word TrolleyAddress) bool isTrolley_INPOS(word TrolleyAddress) bool isTrolley_REGISTRATION(word TrolleyAddress) bool isTrolley_REGVALID(word TrolleyAddress)
60	bool IsTrolleyBrakeOn(word TrolleyAddress)
61	bool IsTrolleyBusy(word TrolleyAddress)
62	bool IsTrolleyDeliveryPending(word TrolleyAddress)
63	bool IsTrolleyInForbiddenBufferSegment(word TrolleyAddress)
64	bool IsTrolleyInForbiddenSegment(word TrolleyAddress)
65	bool IsTrolleyInSegment(word TrolleyAddress, word SegmentNumber)
66	bool IsTrolleyInStopSegment(word TrolleyAddress)
67	bool isTrolleyInSwitch(word TrolleyAddress, dword &SwitchAddress, dword &EntancesCount, dword &ExitsCount)
68	bool isTrolleyInTrouble(word TrolleyAddress)
69	bool isTrolleyMovingToBuffer(word TrolleyAddress)
70	bool IsTrolleyStopped(word TrolleyAddress)
71	bool IsTrolleyStoppedInBuffer(word TrolleyAddress)
72	bool isTrolleyTAEnabled(word TrolleyAddress)
73	bool ManualMode(word TrolleyAddress)
74	bool MessageInProgress(byte DKZAddress)

75	bool MessageSuccess(byte DKZAddress)
76	void MoveTrolley(word TrolleyAddress, word Delta, byte Config, byte Offset)
77	void MoveTrolleyToSegment(word TrolleyAddress, word SegmentNumber, byte Config, byte Offset)
78	void MoveTrolleyToStation(word TrolleyAddress, byte StationNumber, byte StationType, byte Substation, byte Config, byte Offset)
79	byte NumTypeStationLoadUnloadWay(byte StationNumber, byte StationType)
80	bool NumTypeStationNumConvRoutePoint(byte StationNumber, byte StationType, byte SubstationNumber, word &RoutePoint, dword &Position, dword &MachineNumber)
81	void Recirculate(word TrolleyAddress, byte Config, byte Offset)
82	void ResetByPassPositioning(word TrolleyAddress)
83	word RoutingPointZone(word RoutePoint)
84	bool RTEndNextSeg(word TrolleyAddress)
85	bool SendDKZReset(byte DKZAddress)
86	bool SendFakeRoutingTable(word TrolleyAddress, word DefaultSegment)
87	bool SendRouteTable(word TrolleyAddress, word PathIndex, word StartSegment, word EndSegment, byte ConfigByte, byte Offset)
88	bool SendTrolleyReset(word TrolleyAddress)
89	void SetBufferState(word BufferNumber, bool State)
90	void SetByPassPositioning(word TrolleyAddress)
91	void SetDefaultConfig(byte ConfigByte, byte Offset)
92	void SetTrolleyBusy(word TrolleyAddress, bool ---)
93	void SetTrolleyCapacity(word TrolleyAddress, word Capacity)
94	void SetTrolleyCurrentOccupation(word TrolleyAddress, word Occupation)
95	void StartBufferSystem()
96	void Stop(word TrolleyAddress)
97	void StopBufferSystem()

98	void TrolleyReset(word TrolleyAddress)
99	void WMKAutomatic(word WMKAddress)
100	bool WMKErrorContactorCURVE(word WMKAddress) bool WMKErrorContactorSTRAIGHT(word WMKAddress) bool WMKErrorTB4(word WMKAddress) bool WMKErrorTB3(word WMKAddress) bool WMKErrorTB2(word WMKAddress) bool WMKErrorTB1(word WMKAddress) bool WMKNonEQInitiator2(word WMKAddress) bool WMKNonEQInitiator1(word WMKAddress)
101	byte WMKGetControlA(word WMKAddress)
102	byte WMKGetControlB(word WMKAddress)
103	byte WMKGetFlags(word WMKAddress)
104	word WMKGetOffset()
105	bool WMKIsTrolleyInside(word WMKAddress)
106	bool WMKMainSW(word WMKAddress) bool WMKMsgL(word WMKAddress) bool WMKMsgN24(word WMKAddress) bool WMKWorking (word WMKAddress) bool WMKPosition1(word WMKAddress) bool WMKPosition2(word WMKAddress)
107	void WMKMoveCurveManual(word WMKAddress, byte IsolatorBlocksConfig)
108	void WMKMoveCurveManual2(word WMKAddress)
109	void WMKMoveStraightManual(word WMKAddress, byte IsolatorBlocksConfig)
110	void WMKMoveStraightManual2(word WMKAddress)
111	bool WMKPlausibilityError(word WMKAddress) bool WMKTimeoutError(word WMKAddress) bool WMKOutputsLampsOverload(word WMKAddress) bool WMKIsolatorBlockOutputOverload(word WMKAddress) bool WMKInitiatorVoltageSupply(word WMKAddress) bool WMKEmergencyPushButton(word WMKAddress)
112	void WMKReset(word WMKAddress)
113	void WMKSetIsolatorBlock(word WMKAddress, byte IsolatorBlocksConfig)
114	bool WMKStateTB4(word WMKAddress) bool WMKStateTB3(word WMKAddress) bool WMKStateTB2(word WMKAddress) bool WMKStateTB1(word WMKAddress) bool WMKInitiator2(word WMKAddress) bool WMKInitiator1(word WMKAddress)
115	byte WMKStatus(word WMKAddress)
116	void WMKStopMovement(word WMKAddress)
117	bool WMKTimeoutError(word WMKAddress)
118	bool WMKWorking(word WMKAddress)
119	void ZonesSubPaths(word Source, word Target, word &SubPath1, word &SubPath2, word &SubPath3, word &SubPath4, word &SubPath5)

Descripción de los métodos:

1

```
void AddDKZ(dword Id,
            byte DKZAddress)
```

Descripción general

Devuelve el identificado de ese DKZ en el sistema.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `dword Id`
   Identificador del DKZ en el sistema.
2. `byte DKZAddress`
   Dirección Profibus del DKZ dentro de la red de bus de campo.

2

```
void CarriageAutomatic(word TrolleyAddress)
```

Descripción general

Pone el carro en modo automático

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word TrolleyAddress`
   Número del carro que se desea consultar.

3

```
void CarriageBackwardFastManual(word TrolleyAddress)
```

Descripción general

Realiza el movimiento del transportador del carro en velocidad rápida hacia atrás en modo manual.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word TrolleyAddress`
   Número del carro que se desea consultar.

4

```
void CarriageBackwardSlowManual(word TrolleyAddress)
```

Descripción general

Realiza el movimiento del transportador del carro en velocidad lenta hacia atrás en modo manual.

Retorno

void

Este método no retorna ningún valor.

Parámetros

1. word TrolleyAddress
   Número del carro que se desea consultar.

5

```
void CarriageForwardFastManual(word TrolleyAddress)
```

Descripción general

Realiza el movimiento del transportador del carro en velocidad rápida hacia delante en modo manual.

Retorno

void

Este método no retorna ningún valor.

Parámetros

1. word TrolleyAddress
   Número del carro que se desea consultar.

6

```
void CarriageForwardSlowManual(word TrolleyAddress)
```

Descripción general

Realiza el movimiento del transportador del carro en velocidad lenta hacia delante en modo manual.

Retorno

void

Este método no retorna ningún valor.

Parámetros

1. word TrolleyAddress
   Número del carro que se desea consultar.

7

```
bool ConvChainEmpty(word TrolleyAddress)
bool ConvChainFull(word TrolleyAddress)
bool ConvChainRunning(word TrolleyAddress)
bool ConvChainStopped(word TrolleyAddress)
```

Descripción general

Métodos que obtienen la misma información que el anterior, pero consultando un estado determinado.
EMPTY → Transportador vacío.
FULL → Transportador con carga
RUNNING → Transportador con cadenas en movimiento.
STOPPED → Transportador parado entre posiciones.

Retorno

bool
Retorna el valor indicado en cada uno de ellos.
True en caso de estar en el estado indicado y false en caso contrario.

Parámetros

1. word **TrolleyAddress**
   Número del carro que se desea consultar.

8

```
void ConveyorLeftManual(word TrolleyAddress)
```

Descripción general

Realiza el movimiento del transportador del carro hacia la izquierda en modo manual.

Retorno

void
Este método no retorna ningún valor.

Parámetros

1. word TrolleyAddress
   Número del carro que se desea consultar.

9

```
void ConveyorLoadLeft(word TrolleyAddress)
```

Descripción general

Activa la carga del transportador del carro por la parte izquierda.

Retorno

void
Este método no retorna ningún valor.

Parámetros

1. word TrolleyAddress
   Número del carro que se desea consultar.

10

```
void ConveyorLoadRight(word TrolleyAddress)
```

Descripción general

Activa la carga del transportador del carro por la parte derecha.

Retorno

void

Este método no retorna ningún valor.

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

11

```
void ConveyorRightManual(word TrolleyAddress)
```

Descripción general

Realiza el movimiento del transportador del carro hacia la derecha en modo manual.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

12

```
void ConveyorUnLoadleft(word TrolleyAddress)
```

Descripción general

Activa la descarga del transportador del carro por la parte izquierda.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

13

```
void ConveyorUnLoadRight(word TrolleyAddress)
```

Descripción general

Activa la descarga del transportador del carro por la parte derecha.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

14

```
void DisableTrolleyTA(word TrolleyAddress)
```

Descripción general

Deshabilita el carro indicado en el parámetro `TrolleyAddress`, para entrar a formar parte de los que son evaluados como más adecuados para la ejecución de una orden. Esta lógica de búsqueda de carro más adecuado es realizada internamente por el Sistema de Control Galileo.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word TrolleyAddress`
   Número del carro que se desea consultar.

15

```
void DKZAutomatic(byte DKZAddress)
```

Descripción general

Pone el DKZ en modo automático.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `byte DKZAddress`
   Dirección Profibus del DKZ dentro de la red de bus de campo.

16

```
void DKZManual(byte DKZAddress)
```

Descripción general

Pone el DKZ en modo manual.

Retorno

`void`

Este método no retorna nada.

Parámetros

1. `byte DKZAddress`
   Dirección Profibus del DKZ dentro de la red de bus de campo.

17

```
void DKZReset(byte DKZAddress)
```

Descripción general

Resetea el DKZ (por medio de la palabra de control). Acusa todos los errores existentes en un segmento.

Retorno

`void`
Este método no retorna nada.

Parámetros

1. `byte` `DKZAddress`
   Dirección Profibus del DKZ dentro de la red de bus de campo.

18

```
void EnableTrolleyTA(word TrolleyAddress)
```

Descripción general

Habilita el carro indicado en el parámetro `TrolleyAddress`, para entrar a formar parte de los que son evaluados como más adecuados para la ejecución de una orden. Esta lógica de búsqueda de carro más adecuado es realizada internamente por el Sistema de Control Galileo.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

19

```
bool ExistsTrolley(word TrolleyAddress)
```

Descripción general

Comprueba si el carro está registrado en el sistema.

Retorno

`true`
Existe el carro indicado dentro del DKZ.
`false`
No existe el carro dentro del DKZ.

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

20

```
bool Failure (word TrolleyAddress)
```

Descripción general

Comprueba si el carro está informando de un error en su Status Word

Retorno

`true`
El carro informa de un error en su Status Word.
`false`
El carro NO informa de un error

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar si tiene avería.

21

```
bool GetBufferState (word Buffer)
```

Descripción general

Comprueba en el estado del buffer consultado.

Retorno

`true`
Activo (se tiene en cuenta en la asignación de bufferes)
`false`
No activo (excluido)

Parámetros

1. `word` `Buffer`
   Número del buffer que se desea consultar su estado.

22

```
byte GetConveyorChainStatusCode(word TrolleyAddress)
```

Descripción general

Obtiene el estado del transportador de cadenas que se encuentra a bordo del carro(bits de STATUS B)

Retorno

`byte`
Retorna el valor del código de estado del transportador del carro:
`0` → La carga no está posicionada correctamente sobre el transportador.
`1` → El transportador se encuentra en movimiento.
`2` → El transportador está vacío.
`3` → Existe carga en el transportador.

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

23

```
word GetDestinationSegmentFrom(word SegmentNumber,
                               word Delta)
```

Descripción general

Devuelve el segmento que esta a una distancia `Delta` de otro segmento

Retorno

`word`
Número de segmento a la distancia determinada del indicado.

Parámetros

1. `word` `SegmentNumber`

Segmento de inicio que se desea consultar.
2. word Delta
Número de segmentos que desea desplazarse.

24

```
word GetDistanceBetweenSegments(word StartSegment,
                                word EndSegment)
```

Descripción general

Devuelve el número de segmentos que por los que tendrá que pasar un carro para ir del segmento inicial al final.

Retorno

word
Retorna el número de segmentos que existen entre los dos indicados como parámetros.

Parámetros

1. word StartSegment
Segmento inicial para la consulta.
2. word EndSegment
Segmento final para la consulta.

25

```
word GetDistanceBetweenSegments2(word StartSegment,
                                 dword RealStartPosition,
                                 word EndSegment,
                                 dword RealEndPosition)
```

Descripción general

Devuelve el número de segmentos por los que tendrá que pasar un carro para ir del segmento inicial (en la posición inicial) al segmento final (en la posición final).
La diferencia con el método anterior es que este método es más preciso al tener en cuenta las posiciones físicas en milímetros además de los segmentos.

Retorno

word
Retorna el número de segmentos por los que es necesario pasar para ir del inicial al final indicados. A diferencia con el método anterior, al tener en cuenta los milímetros es posible consultar si él se encuentra al principio o final del segmento por ejemplo (lo que supone desplazamiento o no)
Si no es posible establecer una unión entre el segmento inicial y final devuelve el valor -1

Parámetros

1. word StartSegment
Segmento inicial de consulta.
2. dword RealStartPosition
Posición real inicial de consulta.
3. word EndSegment
Segmento final de consulta.
4. dword RealEndPosition

Posición real final para la consulta.

26

```
byte GetDKZCounter()
```

Descripción general

Consulta el número de DKZs configurados en el sistema.

Retorno

byte
Retorna el número de DKZ configurado.

Parámetros

Este método no requiere parámetros de entrada.

27

```
word GetMachineSegment (word MachineNumber)
```

Descripción general

Devuelve el número de segmento en el que se encuentra una máquina. En el archivo de configuración se definen estaciones asociadas a máquinas, cada una de ellas situadas en un segmento. Utiliza esta información para obtener la posición.

Retorno

word
Número de segmento en el que se encuentra la máquina consultada.

Parámetros

1. word MachineNumber
   Número de máquina que se desea consultar su segmento.

28

```
word GetNearestStationToTrolley(word TrolleyAddress)
```

Descripción general

Devuelve el número de máquina (según lo configurado en el fichero .ini) más cercano al carro indicado, teniendo en cuenta las subestaciones.

Retorno

word
Número de máquina más cercana al carro teniendo en cuenta las subestaciones.

Parámetros

2. word TrolleyAddress
   Número de carro que se desea actualizar.

29

```
word GetNearestTrolley(word SegmentNumber)
```

Descripción general

Devuelve el carro más cercano al segmento indicado.

Retorno

word

Retorna el número de carro más cercano al segmento indicado.

Parámetros

1. word SegmentNumber
   Número de segmento que se desea consultar.

30

```
word GetNearestTrolleyToStation(byte StationNumber,
                                byte StationType,
                                byte Substation,
                                bool TraceLog)
```

Descripción general

Devuelve el carro más cercano a una estación dada. El valor -1 indica que no se han encontrado carros.

Retorno

word

Retorna el número de carro más cercano a la estación indicada.

Parámetros

1. byte StationNumber
   Número de estación que se desea consultar.
2. byte StationType
   Tipo de estación que se desea consultar.
3. byte Substation
   Número de subestación que se desea consultar.
4. bool TraceLog
   True: habilita la traza del método en Galileo.log (Info). Solo para pruebas ya que consume muchos recursos.
   False: tiene en cuenta solo los habilitados para el Rail Bus Master

31

```
word GetNearestTrolleyToStationByMachine(dword NearestStation,
                                         bool Search)
```

Descripción general

Obtiene el carro más cercano a una máquina de una estación. Utiliza la información del archivo de configuración.

Retorno

word

Retorna el número de carro más cercano a una máquina de una estación.

Parámetros

1. dword NearestStation
   Número de máquina de la estación (según lo configurado en el fichero .ini), teniendo en cuenta las subestaciones.
2. bool Search
   True: tiene en cuenta todos los trolleys.

False: tiene en cuenta solo los habilitados para el Rail Bus Master

32

```
word GetNearestTrolleyToStationEx(byte StationNumber,
                                  byte StationType,
                                  byte Substation,
                                  byte MinDistance)
```

Descripción general

Variación de la función anterior que permite especificar una distancia mínima. De esta manera se da preferencia a carros que están cerca, pero no tanto como los que están dentro de la distancia mínima.

Retorno

`word`

Retorna el número de carro más cercano a la estación indicada.

Parámetros

1. `byte StationNumber`
   Número de estación que se desea consultar.
2. `byte StationType`
   Tipo de estación que se desea consultar.
3. `byte Substation`
   Número de subestación que se desea consultar.
4. `byte MinDistance`
   Distancia mínima en segmentos que se desea evaluar.

33

```
word GetNearestTrolleyToTrolley (byte TrolleyAddress)
```

Descripción general

Obtiene el carro más cercano a un trolley dado.

Retorno

`word`

Carro más cercano a un trolley dado

Parámetros

1. `word TrolleyAddress`
   Número de carro que se desea consultar.

34

```
word GetNextSegmentWithOffset (word StartSegment,
                               word OffsetPosition)
```

Descripción general

Calcula el segmento que está a una distancia determinada (offset) de uno dado. Como ruta para el cálculo utiliza la ruta de recirculado.

Retorno

`word`

Número de segmento que está a una distancia dada.

Parámetros

1. `word` StartSegment
   Número de segmento a partir del cual se desea consultar.
2. `word` OffsetPosition
   Distancia respecto al segmento inicial indicado.

35

```
word GetSegment(word Position)
```

Descripción general

Consulta el segmento a partir de la posición.

Retorno

`word`
Número de segmento que corresponde a la posición física indicada.

Parámetros

1. `word` Position
   Posición en milímetros que se desea consultar.

36

```
bool GetSegmentData(word SegmentNumber,
                    word &InitPosition,
                    dword &EndPosition)
```

Descripción general

Consulta los datos físicos de un segmento.

Retorno

`true`
Existe el segmento consultado.
`false`
No existe el segmento consultado.

Parámetros

1. `word` SegmentNumber
   Segmento que se desea consultar.
2. `word` InitPosition
   Parámetro de salida que indica la posición física en milímetros de inicio del segmento.
3. `dword` EndPosition
   Parámetro de salida que indica la posición física en milímetros de final del segmento.

37

```
word GetSubStationCount(byte StationNumber,
                        byte StationType)
```

Descripción general

Devuelve el numero de subestaciones que tiene configuradas una determinada estación.

Retorno

word

Retorna el número de subestaciones que tiene la estación indicada.

Parámetros

1. byte StationNumber
   Número de estación que se desea consultar.
2. byte StationType
   Tipo de estación que se desea consultar.

38

```
word GetTrolleyCapacity(word TrolleyAddress)
```

Descripción general

Obtiene la capacidad máxima del carro. El componente no mantiene esta ocupación, simplemente devuelve la cargada.

Retorno

word

Retorna la capacidad máxima del carro.

Parámetros

1. word TrolleyAddress
   Número de carro que se desea consultar.

39

```
byte GetTrolleyControlA (word TrolleyAddress)
```

Descripción general

Devuelve la ControlWord A de un carro determinado. Se obtiene desde la imagen de proceso del DKZ

Retorno

byte

ControlWord A de un carro determinado

Parámetros

1. word TrolleyAddress
   Número de carro que se desea consultar.

40

```
byte GetTrolleyControlB (word TrolleyAddress)
```

Descripción general

Devuelve la ControlWord B de un carro determinado. Se obtiene desde la imagen e proceso del DKZ

Retorno

byte

ControlWord B de un carro determinado

Parámetros

1. word TrolleyAddress
Número de carro que se desea consultar.

41

```
word GetTrolleyCurrentOccupation(word TrolleyAddress)
```

Descripción general

Obtiene la ocupación actual del carro. El componente almacena esta ocupación, simplemente devuelve la indicada por control.

Retorno

word
Retorna el número de trackings que tiene el carro.

Parámetros

2. word TrolleyAddress
Número de carro que se desea consultar.

42

```
word GetTrolleyCurrentSpeed(word TrolleyAddress)
```

Descripción general

Obtiene la velocidad (calculada) de un carro determinado. La velocidad se calcula en función de la posición indicada por el carro y el lapso de tiempo entre ciclos

Retorno

word
Devuelve la velocidad (calculada) de un carro determinado

Parámetros

1. word TrolleyAddress
Número de carro que se desea consultar.

43

```
void GetTrolleyDestinationBuffer (word TrolleyAddress,
                                  word &SegmentBuffer,
                                  dword &Position)
```

Descripción general

Devuelve en un parámetro por referencia el buffer (y su posición) que se ha asignado a este trolley.

Retorno

void
Este método no retorna ningún valor.

Parámetros

1. word TrolleyAddress
Número de carro a consultar
2. word &SegmentBuffer
Número del segmento
3. dword &Position

Valor de la posición absoluta en mm

44

```
dword GetTrolleyDistanceToPos (word TrolleyAddress,
                               word Segment,
                               dword Position)
```

Descripción general

Devuelve la distancia en segmentos desde la posición actual del trolley a una posición (segmento, posición) determinada

Retorno

`dword`

Distancia en segmentos

Parámetros

1. `word TrolleyAddress`
   Número de carro a consultar
2. `word Segment`
   Número del segmento
3. `dword Position`
   Valor de la posición absoluta en mm

45

```
byte GetTrolleyDKZ(word TrolleyAddress)
```

Descripción general

Comprueba en que DKZ se ha realizado el registro del carro indicado.

Retorno

`byte`

Dirección Profibus del DKZ donde se ha registrado el carro indicado.

Parámetros

1. `word TrolleyAddress`
   Número del carro que se desea consultar.

46

```
byte GetTrolleyErrorNumber (word TrolleyAddress)
```

Descripción general

Devuelve el código de error de un trolley determinado. Este código de error se obtiene desde la imagen de proceso del carro.

Retorno

`byte`

Código de error del trolley indicado.

Parámetros

1. `word TrolleyAddress`
   Número del carro que se desea consultar.

47

```
byte GetTrolleyFlags(word TrolleyAddress)
```

Descripción general

Obtiene el byte de flags del carro.
(`byte` *Flags* del información cíclica del carro)

Retorno

`byte`
Retorna los flags de información del carro indicado.

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

48

```
word GetTrolleyInTroubleWith (word TrolleyAddress)
```

Descripción general

Devuelve la dirección de carro que está bloqueando al pasado como parámetro

Retorno

`word`
Número de carro que está bloqueando al pasado como parámetro

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

49

```
dword GetTrolleyPosition(word TrolleyAddress)
```

Descripción general

Consulta la posición del vehículo en milímetros (bits de *STATUS B+STATUS C+STAUTS D*)

Retorno

`dword`
Valor de la posición actual del carro en milímetros.

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

50

```
word GetTrolleyRealSegment(word TrolleyAddress)
```

Descripción general

Consulta el segmento real basado en la posición física del carro.

Retorno

`word`

Número de segmento en el que se encuentra el carro indicado.

Parámetros

1. `word TrolleyAddress`
   Número de carro que se desea consultar.

51

```
word GetTrolleyRoutePoint(word TrolleyAddress)
```

Descripción general

Obtiene el punto en el que se encuentra el carro indicado.
(palabra *PosPoint* de la información cíclica del carro)

Retorno

`word`
Retorna el punto en el que se encuentra el carro indicado.

Parámetros

1. `word TrolleyAddress`
   Número del carro que se desea consultar.

52

```
byte GetTrolleyStatusCode(word TrolleyAddress)
```

Descripción general

Obtiene el estado del carro dado por la *STATUS B*.

Retorno

`byte`
Retorna el valor del código de estado de carro:

a. `0` → Carro en servicio
b. `1` → Carro moviéndose, y finalización de su tabla de ruta en el siguiente segmento al que se encuentra actualmente.
c. `2` → Carro parado debido a que existen errores en la ruta.
d. `3` → Carro parado por cualquier otro error que no sea la ruta.

Parámetros

1. `word TrolleyAddress`
   Número del carro que se desea consultar.

53

```
word GetTrolleyTroubleDistance(word TrolleyAddress)
```

Descripción general

Devuelve la distancia en segmentos a la que esta del carro con el que se encuentra en conflicto.

Retorno

`word`
Número de segmentos que existen entre ambos carros.

Parámetros

1. word TrolleyAddress
Número del carro que se desea consultar.

54

```
void Init()
```

Descripción general

Inicializa completamente la electrovía de LJU.

Retorno

void
Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

55

```
bool InPosition (word TrolleyAddress)
```

Descripción general

Devuelve cierto si el carro ha activado el bit "VEH in position"

Retorno

true
El carro consultado tiene el bit de "In position" activo
false
No tiene el bit de "In position" activo.

Parámetros

1. word TrolleyAddress
Número del carro que se desea consultar.

56

```
void InvalidateMovements(word TrolleyAddress)
```

Descripción general

Invalida en el carro seleccionado las tablas de rutas actuales.

Retorno

void
Este método no retorna ningún valor.

Parámetros

1. word TrolleyAddress
Número de carro al cual se le invalidarán los movimientos.

57

```
bool IsAllowedLoad (dword UnloadMachine,
                    dword LoadMachine)
```

Descripción general

Devuelve cierto si es posible cargar en la máquina "LoadMachine" tras descarga en la máquina "UnloadMachine"

Retorno

`true`
La carga se permite
`false`
La carga no se permite

Parámetros

1. `dword UnloadMachine`
   Número de máquina en la que se realizará la descarga previa
2. `dword LoadMachine`
   Número de máquina donde se desea cargar con la segunda orden

58

```
bool IsASwitch (word aSegment,
                dword &SwitchAddress,
                dword &EntancesCount,
                dword &ExitsCount)
```

Descripción general

Devuelve cierto si el segmento indicado es parte de un desvío. En los parámetros por referencia se devuelven los datos del desvío

Retorno

`true`
El segmento indicado se ha definido como parte de un desvío en el archivo de configuración
`false`
El segmento indicado no forma parte de ningún desvío

Parámetros

1. `word aSegment`
   Número de segmento a consultar
2. `dword &SwitchAddress`
   Dirección del desvío al que pertenece el segmento
3. `dword &EntrancesCount`
   Número de entradas del desvío al que pertenece el segmento
4. `dword &ExitsCount`
   Número de salidas del desvío al que pertenece el segmento

59

```
bool isTrolley_NOCOMM(word TrolleyAddress)
bool isTrolley_MANUALMODE(word TrolleyAddress)
bool isTrolley_ERROR(word TrolleyAddress)
bool isTrolley_INPOS(word TrolleyAddress)
bool isTrolley_REGISTRATION(word TrolleyAddress)
bool isTrolley_REGVALID(word TrolleyAddress)
```

Descripción general

Estos métodos devuelven el valor de cada uno de los bits definidos en el byte *STATUS A* del carro:
NOCOM → Estado de la comunicación con el carro.
MANUALMODE → Carro en modo manual.
ERROR → Carro en error.
INPOS → Carro situado en su posición destino
REGISTRATION → Carro dado de alta en un DKZ
REGVALID → Carro registrado correctamente.

Retorno

bool
Retorna el estado del bit que indica cada uno.

Parámetros

2. word TrolleyAddress
   Número del carro que se desea consultar.

60

```
bool IsTrolleyBrakeOn (word TrolleyAddress)
```

Descripción general

Devuelve cierto si carro consultado ha activado el freno. Esta información se obtiene desde la imagen de proceso

Retorno

true
El carro consultado tiene activo el bit de freno
false
El carro no tiene activo el bit de freno

Parámetros

1. word TrolleyAddress
   Número del carro que se desea consultar.

61

```
bool IsTrolleyBusy (word TrolleyAddress)
```

Descripción general

Devuelve cierto si el carro ha sido marcado por el programa de control como "Carro ocupado". Los carros ocupados no son utilizados en el proceso de asignación a buffer

Retorno

true
El carro ha sido marcado previamente como "carro ocupado"
false
El carro no está marcado como "carro ocupado"

Parámetros

1. word TrolleyAddress
   Número del carro que se desea consultar.

62

```
bool IsTrolleyDeliveryPending (word TrolleyAddress)
```

Descripción general

Devuelve cierto si el carro tiene tablas de ruta pendientes de enviar

Retorno

`true`
El carro tiene una tabla de ruta pendiente, pero aun no la ha enviado
`false`
El carro no tiene mensajes pendientes

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

63

```
bool IsTrolleyInForbiddenBufferSegment (word TrolleyAddress)
```

Descripción general

Devuelve cierto si el carro está actualmente en un segmento configurado como "prohibido para el sistema de buffers". Estos segmentos se configuran en la sección [ForbiddenBufferSegments] y permiten optimizar la gestión de buffers

Retorno

`true`
El carro consultado está en segmento configurado como "prohibido para el sistema de buffers"
`false`
El carro no está en un segmento configurado como "prohibido para el sistema de buffers"

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

64

```
bool IsTrolleyInForbiddenSegment (word TrolleyAddress)
```

Descripción general

Devuelve cierto si el carro está actualmente en un segmento configurado como "prohibido". Estos segmentos se configuran en la sección [ForbiddenSegments] y permiten ayudar al programa de control a gestionar zonas problemáticas donde el envío de una tabla de rutas podría llegar tarde

Retorno

`true`
El carro consultado está en segmento configurado como "prohibido"
`false`
El carro no está en un segmento configurado como "prohibido"

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

65

```
bool isTrolleyInSegment(word TrolleyAddress,
                        word SegmentNumber)
```

Descripción general

Indica si un carro está en un segmento determinado.

Retorno

`true`
El carro indicado se encuentra en el segmento indicado.
`false`
El carro no está en el segmento de consulta.

Parámetros

1. `word TrolleyAddress`
   Número de carro que se desea consultar.
2. `word SegmentNumber`
   Número de segmento que se desea consultar.

66

```
bool IsTrolleyInStopSegment (word TrolleyAddress)
```

Descripción general

Devuelve cierto si el carro está actualmente en un segmento configurado como "de parada". Estos segmentos se configuran en la sección [StopSegments] y permiten ayudar al programa de control a gestionar zonas problemáticas donde el envío de una tabla de rutas podría llegar tarde

Retorno

`true`
El carro consultado está en segmento configurado como "de parada"
`false`
El carro no está en un segmento configurado como "parada"

Parámetros

1. `word TrolleyAddress`
   Número de carro que se desea consultar.

67

```
bool isTrolleyInSwitch (word TrolleyAddress,
                        dword &SwitchAddress,
                        dword &EntancesCount,
                        dword &ExitsCount)
```

Descripción general

Devuelve cierto si el carro consultado está en un segmento que es parte de un desvío. En los parámetros por referencia se devuelven los datos del desvío

Retorno

`true`
El carro indicado se encuentra en un segmento perteneciente a un desvío
`false`
El carro indicado no está en un segmento perteneciente a un desvío

Parámetros

1. `word TrolleyAddress`
   Dirección de carro a consultar
2. `dword &SwitchAddress`
   Dirección del desvío al que pertenece el segmento
3. `dword &EntrancesCount`
   Número de entradas del desvío al que pertenece el segmento
4. `dword &ExitsCount`
   Número de salidas del desvío al que pertenece el segmento

68

```
bool isTrolleyInTrouble(word TrolleyAddress)
```

Descripción general

Comprueba si un carro está en la trayectoria de otro, de manera que le pueda molestar para completar su ruta.

Retorno

`true`
El carro indicado se encuentra en la misma ruta que otro.
`false`
El carro indicado NO está en conflicto con ningún otro.

Parámetros

1. `word TrolleyAddress`
   Número del carro que se desea consultar.

69

```
bool isTrolleyMovingToBuffer (word TrolleyAddress)
```

Descripción general

Devuelve cierto si el carro consultado ha sido seleccionado por el sistema para iniciar un movimiento a buffer

Retorno

`true`
El carro indicado se encuentra moviéndose hacia el buffer.
`false`
El carro indicado NO se encuentra moviéndose hacia el buffer.

Parámetros

1. `word TrolleyAddress`
   Número del carro que se desea consultar.

70

```
bool IsTrolleyStopped (word TrolleyAddress)
```

Descripción general

Devuelve cierto si el carro está parado por estar bloqueado por otro carro

Retorno

`true`
El carro está parado por distancia
`false`
El carro no está parado por distancia

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

71

```
bool IsTrolleyStoppedInBuffer (word TrolleyAddress)
```

Descripción general

Comprueba si un carro está parado en el buffer que tiene asignado.

Retorno

`true`
El carro indicado se encuentra parado en un buffer.
`false`
El carro indicado NO está parado en un buffer.

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

72

```
bool isTrolleyTAEnabled(word TrolleyAddress)
```

Descripción general

Consulta si un carro entra en las búsquedas del componente para encontrar el más cercano a un punto (lógica realizada por el Sistema de Control Galileo).

Retorno

`true`
El carro indicado está habilitado y puede entrar a formar parte de la búsqueda.
`false`
El carro indicado NO está habilitado.

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

73

```
bool ManualMode (word TrolleyAddress)
```

Descripción general

Comprueba si un carro está en modo manual

Retorno

`true`
El carro indicado está en modo manual
`false`
El carro indicado NO está en modo manual.

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

74

```
bool MessageInProgress (byte DKZAddress)
```

Descripción general

Devuelve cierto si el DKZ consultado ha enviado un mensaje DPV-1 y aún no ha obtenido contestación

Retorno

`true`
El mensaje ha sido enviado y aún no se ha recibido respuesta
`false`
No hay mensajes pendientes de respuesta

Parámetros

1. `word` `DKZAddress`
   Número de DKZ sobre el que se consulta

75

```
bool MessageSuccess(byte DKZAddress)
```

Descripción general

Comprueba si la última operación de mensajería tuvo éxito.

Retorno

`true`
La operación se ha realizado con éxito.
`false`
La operación no ha finalizado correctamente.

Parámetros

1. `byte` `DKZAddress`
   Dirección Profibus del DKZ dentro de la red de bus de campo.

76

```
void MoveTrolley(word TrolleyAddress,
                 word Delta,
                 byte Config,
                 byte Offset)
```

Descripción general

Envía la/s ruta/s necesaria/s para mover un carro un número de segmentos determinado.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word TrolleyAddress`
   Número de carro que se desea mover.
2. `word Delta`
   Número de segmentos que se desea mover.
3. `byte ConfigByte`
   Byte de configuración (el byte de configuración indica en que parte del segmento destino se va a realizar la parada del carro, consultar si se requiere los manuales existentes de la configuración de la electrovía LJU) de la ruta que se va a enviar.
4. `byte Offset`
   Valor del offset (el offset indica el punto relativo en el que va a parar el carro con respecto al punto definido en el parámetro `ConfigByte`, consultar si se requiere los manuales existentes de la configuración de la electrovía LJU) que se enviará en la ruta.

77

```
void MoveTrolleyToSegment(word TrolleyAddress,
                          word SegmentNumber,
                          byte Config,
                          byte Offset)
```

Descripción general

Envia la/s ruta/s necesaria/s para mover un carro hasta un segmento determinado.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word TrolleyAddress`
   Número de carro que se desea mover.
2. `word SegmentNumber`
   Número de segmento destino al que se desea desplazar.
3. `byte ConfigByte`
   Byte de configuración (el byte de configuración indica en que parte del segmento destino se va a realizar la parada del carro, consultar si se requiere los manuales existentes de la configuración de la electrovía LJU) de la ruta que se va a enviar.
4. `byte Offset`
   Valor del offset (el offset indica el punto relativo en el que va a parar el carro con respecto al punto definido en el parámetro `ConfigByte`, consultar si se requiere los manuales existentes de la configuración de la electrovía LJU) que se enviará en la ruta.

78

```
void MoveTrolleyToStation(word TrolleyAddress,
                          byte StationNumber,
                          byte StationType,
                          byte Substation,
                          byte Config,
                          byte Offset)
```

Descripción general

Envía la/s ruta/s necesaria/s para mover un carro hasta una estación determinada.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word TrolleyAddress`
   Número de carro que se desea mover.
2. `byte StationNumber`
   Número de estación destino al que se desea desplazar.
3. `byte StationType`
   Tipo de estación destino a la que se desea mover.
4. `byte Substation`
   Número de subestación destino a la que se desea mover.
5. `byte ConfigByte`
   Byte de configuración (el byte de configuración indica en que parte del segmento destino se va a realizar la parada del carro, consultar si se requiere los manuales existentes de la configuración de la electrovía LJU) de la ruta que se va a enviar.
6. `byte Offset`
   Valor del offset (el offset indica el punto relativo en el que va a parar el carro con respecto al punto definido en el parámetro `ConfigByte`, consultar si se requiere los manuales existentes de la configuración de la electrovía LJU) que se enviará en la ruta.

79

```
byte NumTypeStationLoadUnloadWay(byte StationNumber,
                                 byte StationType)
```

Descripción general

Obtiene el modo de carga y el sentido de una estación.

Retorno

`byte`

Retorna si es modo carga o descarga y el sentido:

a. `1` → Load on left way
b. `2` → Load on right way
c. `3` → Unload on left way
d. `4` → Unload on right way
e. `5` → Load & Unload left
f. `6` → Load & Unload right

Parámetros

1. byte StationNumber
Número de estación que se desea consultar.
2. byte StationType
Tipo de estación que se desea consultar.

80

```
bool NumTypeStationNumConvRoutePoint(byte StationNumber,
                                     byte StationType,
                                     byte SubstationNumber,
                                     word &RoutePoint,
                                     dword &Position,
                                     dword &MachineNumber)
```

Descripción general

Convierte un número y tipo de estación determinado en un punto de ruta.

Retorno

true
Parámetro válidos
false
Se le ha pasado el parámetro de subestación erróneo.

Parámetros

1. byte StationNumber
Número de estación que se desea consultar.
2. byte StationType
Tipo de estación que se desea consultar.
3. byte SubstationNumber
Número de subestación que se desea consultar.
4. word RoutePoint
Parámetro de salida que indica el punto de la ruta que corresponde a la estación indicada en los parámetros de entrada.
5. dword Position
Parámetro de salida que indica la posición de la estación indicada en los parámetros de entrada.
6. dword MachineNumber
Parámetro de salida que indica el número de máquina de la estación indicada en los parámetros de entrada

81

```
void Recirculate(word TrolleyAddress,
                 byte Config,
                 byte Offset)
```

Descripción general

Envía la/s ruta/s necesaria/s para hacer recircular a un carro, moviéndolo a través de la electrovía hasta el punto desde el que partió.

Retorno

void
Este método no retorna ningún valor.

Parámetros

1. `word` `TrolleyAddress`
   Número de carro que se desea recircular.
2. `byte` `ConfigByte`
   Byte de configuración (el byte de configuración indica en que parte del segmento destino se va a realizar la parada del carro, consultar si se requiere los manuales existentes de la configuración de la electrovía LJU) de la ruta que se va a enviar.
3. `byte` `Offset`
   Valor del offset (el offset indica el punto relativo en el que va a parar el carro con respecto al punto definido en el parámetro `ConfigByte`, consultar si se requiere los manuales existentes de la configuración de la electrovía LJU) que se enviará en la ruta.

82

```
void ResetByPassPositioning(word TrolleyAddress)
```

Descripción general

Desactiva el bit de bypass del carro necesario para realizar un correcto movimiento. **NOTA**: Actualmente es la propia lógica de la electrovía LJU la que realiza el control de dicho bit.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

83

```
word RoutingPointZone(word RoutePoint)
```

Descripción general

Consulta la zona a la que pertenece un determinado punto de la ruta.

Retorno

`word`
Devuelve la zona a la que pertenece el punto de la ruta que se indica.

Parámetros

2. `word` `RoutePoint`
   Punto de la ruta del que se desea consultar su zona.

84

```
bool RTEndNextSeg(word TrolleyAddress)
```

Descripción general

Comprueba en el estado del carro, si la tabla de ruta finaliza en el próximo segmento (bits *STATUS B*).

Retorno

`true`
La tabla de ruta del carro finaliza en el siguiente segmento al actual.
`false`
La tabla de ruta del carro NO finaliza en el siguiente segmento al actual.

Parámetros

1. `word` `TrolleyAddress`
   Número del carro que se desea consultar.

85

```
bool SendDKZReset(byte DKZAddress)
```

Descripción general

Resetea el DKZ (por medio de un mensaje). Este método además resetea todos los dispositivos conectados a él (desvíos y carros).

Retorno

`true`
Si se ha podido enviar el mensaje (aunque no haya llegado al DKZ).
`false`
NO se ha podido enviar el mensaje.

Parámetros

1. `byte` `DKZAddress`
   Dirección Profibus del DKZ dentro de la red de bus de campo.

86

```
bool SendFakeRoutingTable(word TrolleyAddress,
                          word DefaultSegment)
```

Descripción general

Envía una tabla de rutas "falsa", para forzar el carro a tener una almacenada y no dar un error por estar vacía.

Retorno

`true`
Ha finalizado con éxito.
`false`
NO ha finalizado con éxito.

Parámetros

1. `word` `TrolleyAddress`
   Número de carro al cual se envía la ruta.
2. `word` `DefaultSegment`
   Número del segmento que se envía.

87

```
bool SendRouteTable(word TrolleyAddress,
                    word PathIndex,
                    word StartSegment,
                    word EndSegment,
```

```
                    byte ConfigByte,
                    byte Offset)
```

Descripción general

Envía una tabla de rutas al carro. Los datos se obtienen a partir de los parámetros del fichero .ini configurado desde control.

Retorno

`true`
Se ha podido enviar la tabla de rutas (no tiene por qué haber llegado al carro).
`false`
NO se ha podido enviar la tabla de rutas

Parámetros

1. `word TrolleyAddress`
   Número del carro que se desea consultar.
2. `word PathIndex`
   Índice del camino que se desea seguir.
3. `word StartSegment`
   Segmento inicial de la ruta que se va a enviar.
4. `word EndSegment`
   Segmento final de la ruta que se va a enviar.
5. `byte ConfigByte`
   Byte de configuración (el byte de configuración indica en que parte del segmento destino se va a realizar la parada del carro, consultar si se requiere los manuales existentes de la configuración de la electrovía LJU) de la ruta que se va a enviar.
6. `byte Offset`
   Valor del offset (el offset indica el punto relativo en el que va a parar el carro con respecto al punto definido en el parámetro `ConfigByte`, consultar si se requiere los manuales existentes de la configuración de la electrovía LJU) que se enviará en la ruta.

88

```
bool SendTrolleyReset(word TrolleyAddress)
```

Descripción general

Realiza el reset del carro mediante un mensaje.

Retorno

`true`
Se ha podido enviar el mensaje correctamente (aunque no haya llegado al carro).
`false`
NO se ha podido enviar el mensaje.

Parámetros

1. `word TrolleyAddress`
   Número del carro que se desea consultar.

89

```
void SetBufferState (word BufferNumber
                     bool State)
```

Descripción general

Permite establecer el estado de un buffer, de manera que pueda ser utilizado por el sistema de buffers

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word BufferNumber`
   Número de buffer a consultar
2. `bool State`
   a. Cierto → El buffer está activo
   b. Falso → El buffer está desactivado

90

```
void SetByPassPositioning(word TrolleyAddress)
```

Descripción general

Activa el bit de bypass del carro necesario para realizar un correcto movimiento.
NOTA: Actualmente es la propia lógica de la electrovía LJU la que realiza el control de dicho bit.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word TrolleyAddress`
   Número del carro que se desea consultar.

91

```
void SetDefaultConfig(byte ConfigByte,
                      byte Offset)
```

Descripción general

Establece los parámetros `Config` y `Offset` a utilizar en el movimiento a posición cero.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `byte ConfigByte`
   Byte de configuración (el byte de configuración indica en que parte del segmento destino se va a realizar la parada del carro, consultar si se requiere los manuales existentes de la configuración de la electrovía LJU) de la ruta que se va a enviar.
2. `byte Offset`
   Valor del offset (el offset indica el punto relativo en el que va a parar el carro con respecto al punto definido en el parámetro `ConfigByte`, consultar si se requiere los manuales existentes de la configuración de la electrovía LJU) que se enviará en la ruta.

92

```
void SetTrolleyBusy (word TrolleyAddress,
                     bool Busy)
```

Descripción general

Permite marcar a un carro como ocupado, de manera que no se incluya en el sistema de buffers

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word TrolleyAddress`
   Número de carro que se desea actualizar.
2. `bool Busy`
   a. Cierto → El carro se marca como ocupado
   b. Falso → El carro se marca como no ocupado

93

```
void StartBufferSystem ()
```

Descripción general

Habilita la gestión de buffers, de manera que en cada ciclo de programa se calcula que carros se deben mover a buffer e inicia el movimiento si es necesario.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

94

```
void SetTrolleyCapacity(word TrolleyAddress,
                        word Capacity)
```

Descripción general

Establece la capacidad del carro. Esta capacidad solo se almacena para que la consulte el programa de control. El componente no realiza ninguna operativa con ella

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word TrolleyAddress`
   Número de carro que se desea actualizar.
2. `word Capacity`
   Valor de la capacidad que se desea establecer.

95

```
void SetTrolleyCurrentOccupation(word TrolleyAddress,
                                 word Occupation)
```

Descripción general

Establece la ocupación del carro. Esta ocupación solo se almacena para que la consulte el programa de control. El componente no realiza ninguna operativa con ella.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word TrolleyAddress`
   Número de carro que se desea actualizar.
2. `word Occupation`
   Valor de la ocupación que se desea establecer.

96

```
void Stop(word TrolleyAddress)
```

Descripción general

Detiene el carro indicado en el parámetro `TrolleyAddress`.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `word TrolleyAddress`
   Número del carro que se desea consultar.

97

```
void StopBufferSystem ()
```

Descripción general

Deshabilita la gestión de buffer. El componente ya no intentará asignar carros a buffers

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

98

```
void TrolleyReset (word TrolleyAddress)
```

Descripción general

Resetea las averías del carro indicado

Retorno

void
Este método no retorna ningún valor.

Parámetros

1. word TrolleyAddress
   Número del carro que se desea resetear.

99

```
void WMKAutomatic(word WMKAddress)
```

Descripción general

Sitúa el desvío en modo automático.

Retorno

void
Este método no retorna ningún valor.

Parámetros

1. word WMKAddress
   Número de desvío que se desea consultar.

100

```
bool WMKErrorContactorCURVE(word WMKAddress)
bool WMKErrorContactorSTRAIGHT(word WMKAddress)
bool WMKErrorTB4(word WMKAddress)
bool WMKErrorTB3(word WMKAddress)
bool WMKErrorTB2(word WMKAddress)
bool WMKErrorTB1(word WMKAddress)
bool WMKNonEQInitiator2(word WMKAddress)
bool WMKNonEQInitiator1(word WMKAddress)
```

Descripción general

Estos métodos devuelven el valor de los bits definidos en el byte *STATUS C* del desvío (consultar documentación técnica de la electrovía LJU).
ErrorContactorCURVE → Error en el contactor del movimiento a curva
ErrorContactorSTRAIGHT → Error en el contactor del movimiento a recta.
ErrorTB4….TB1 → Error en el contactor que controla el Isolator Block 1…4.
NonEQInitiator2 (curva) → Se ha producido un error en el accionador al realizar el movimiento a curva. El accionador 2 sólo es para curva.
NonEQInitiator1 (recta) → Se ha producido un error en el accionador al realizar el movimiento a recta. El accionador 1 sólo es para recta.

Retorno

bool
Retorna el estado del bit que se indica en el nombre del método correspondiente.

Parámetros

1. word WMKAddress
   Número de desvío que se desea consultar.

101

```
byte WMKGetControlA (word WMKAddress)
```

Descripción general

Obtiene el valor de la Control Word A del desvío indicado. Es importante tener en cuenta que el valor de Control Word **antes** de escribirla.

Retorno

byte
Retorna el valor de la Control Word A del desvío

Parámetros

1. word WMKAddress
   Número de desvío que se desea consultar.

102

```
byte WMKGetControlB (word WMKAddress)
```

Descripción general

Obtiene el valor de la Control Word B del desvío indicado. Es importante tener en cuenta que el valor de Control Word **antes** de escribirla.

Retorno

byte
Retorna el valor de la Control Word B del desvío

Parámetros

1. word WMKAddress
   Número de desvío que se desea consultar.

103

```
byte WMKGetFlags(word WMKAddress)
```

Descripción general

Consulta el byte de flags del desvío indicado. Esta información no se refresca todos los ciclos, sino que es actualizada cada vez que se consulta dicho desvío, pasando siempre por todos y cada uno de los desvíos y los carros.
Ejem: Si existiesen tres desvíos y tres carros, la información sería actualizada cada seis bloques de datos.

Retorno

byte
Retorna el valor del flag del desvío indicado.

Parámetros

2. word WMKAddress
   Número de desvío que se desea consultar.

104

```
word WMKGetOffset ()
```

Descripción general

Obtiene el offset configurado el archivo de configuración para las direcciones de los desvíos

Retorno

word

Este método retorna el offset que se ha configurado en el archivo de configuración para las direcciones de los desvíos

Parámetros

Este método no requiere parámetros de entrada.

105

```
bool WMKIsTrolleyInside (word WMKAddress)
```

Descripción general

Devuelve cierto si algún carro está atravesando en este momento el desvío consultado

Retorno

bool

Retorna cierto si el desvío está siendo atravesado por un carro

Parámetros

1. word WMKAddress
   Número de desvío que se desea consultar.

106

```
bool WMKMainSW(word WMKAddress)
bool WMKMsgL(word WMKAddress)
bool WMKMsgN24(word WMKAddress)
bool WMKWorking (word WMKAddress)
bool WMKPosition1(word WMKAddress)
bool WMKPosition2(word WMKAddress)
```

Descripción general

Estos métodos devuelven el valor de los bits definidos en el byte *STATUS A* del desvío.
MainSW → Desvío con tensión.
MsgL → Error en la conexión de fases del desvío.
MsgN24 → Error en la conexión de 24V del desvío.
Working → El desvío está cambiando de posición.
Position1 → El desvío se encuentra en la izquierda.
Position2 → El desvío se encuentra en la derecha.

Retorno

bool

Retorna el valor del estado indicado por el método.

Parámetros

1. word WMKAddress
Número de desvío que se desea consultar.

107

```
void WMKMoveCurveManual(word WMKAddress
                        byte IsolatorBlocksConfig)
```

Descripción general

Sitúa el desvío en modo manual en la posición de curva. Permite establecer la configuración de los "IsolatorsBlocks" de manera explícita.

Retorno

void
Este método no retorna ningún valor.

Parámetros

1. word WMKAddress
Número de desvío que se desea activar.
2. byte IsolatorBlocksConfig
Configuración de los "IsolatorBlocks". Consultar manual de LJU acerca de "Control command A: Points controller isolator block configuration"

108

```
void WMKMoveCurveManual2(word WMKAddress)
```

Descripción general

Sitúa el desvío en modo manual en la posición de curva. No modifica la configuración de los "isolator blocks"

Retorno

void
Este método no retorna ningún valor.

Parámetros

1. word WMKAddress
Número de desvío que se desea activar.

109

```
void WMKMoveStraightManual(word WMKAddress
                           byte IsolatorBlocksConfig)
```

Descripción general

Sitúa el desvío en modo manual en la posición de recta. Permite establecer la configuración de los "IsolatorsBlocks" de manera explícita.

Retorno

void
Este método no retorna ningún valor.

Parámetros

1. word WMKAddress
Número de desvío que se desea activar.

2. byte IsolatorBlocksConfig
Configuración de los "IsolatorBlocks". Consultar manual de LJU acerca de "Control command A: Points controller isolator block configuration"

110

```
void WMKMoveStraightManual2 (word WMKAddress)
```

Descripción general

Sitúa el desvío en modo manual en la posición de recta. No modifica la configuración de los "isolator blocks"

Retorno

void
Este método no retorna ningún valor.

Parámetros

1. word WMKAddress
Número de desvío que se desea activar.

111

```
bool WMKPlausibilityError(word WMKAddress)
bool WMKTimeoutError(word WMKAddress)
bool WMKOutputsLampsOverload(word WMKAddress)
bool WMKIsolatorBlockOutputOverload(word WMKAddress)
bool WMKInitiatorVoltageSupply(word WMKAddress)
bool WMKEmergencyPushButton(word WMKAddress)
```

Descripción general

Estos métodos devuelven el valor de los bits definidos en el byte *STATUS B* del desvío.
PlausibilityError → Error de plausibidad (por ejemplo, si están accionados los sensores tanto de curva como de recta al mismo tiempo).
TimeoutError → Timeout en realizar el movimiento para cambio de posición de recta a curva y viceversa.
OutputsLampsOverload → Sobretensión en las salidas digitales (consultar documentación técnica de la electrovía LJU).
IsolatorBlockOutputOverload → Sobretensión en Isolator Blocks (consultar documentación técnica de la electrovía LJU).
InitiatorVoltageSupply → Caída de tensión en Initiator (consultar documentación técnica de la electrovía LJU).
EmergencyPushButton → Accionada la seta/s de seguridad.

Retorno

bool
Retorna el estado del bit que se indica en el nombre del método correspondiente.

Parámetros

1. word WMKAddress
Número de desvío que se desea consultar.

112

```
void WMKReset(word WMKAddress)
```

Descripción general

Resetea el posible error el desvío indicado.

Retorno

void
Este método no retorna ningún valor.

Parámetros

1. word WMKAddress
   Número de desvío que se desea consultar.

113

```
void WMKSetIsolatorBlock (word WMKAddress
                          byte IsolatorBlocksConfig)
```

Descripción general

Establece la configuración de los "isolator blocks" para un desvío determinado

Retorno

void
Este método no retorna ningún valor.

Parámetros

1. word WMKAddress
   Número de desvío que se desea activar.
2. byte IsolatorBlocksConfig
   Configuración de los "IsolatorBlocks". Consultar manual de LJU acerca de "Control command A: Points controller isolator block configuration"

114

```
bool WMKStateTB4(word WMKAddress)
bool WMKStateTB3(word WMKAddress)
bool WMKStateTB2(word WMKAddress)
bool WMKStateTB1(word WMKAddress)
bool WMKInitiator2(word WMKAddress)
bool WMKInitiator1(word WMKAddress)
```

Descripción general

Estos métodos devuelven el valor de los bits definidos en el byte *STATUS D* del desvío (consultar la documentación técnica de la electrovía LJU).
StateTB4…TB1 → Estado en el que se encuentran los Isolator Block 1…4 (1 están correctos y 0, existe algún problema).
Initiator2 (curva) → El accionador está en la posición correcta, curva.
Initiator1 (recta) → El accionador está en la posición correcta, recta.

Retorno

bool
Retorna el estado del bit que se indica en el nombre del método correspondiente.

Parámetros

1. word WMKAddress
Número de desvío que se desea consultar.

115

```
byte WMKStatus(word WMKAddress)
```

Descripción general

Obtiene el estado del desvío (bits de *STATUS B*)

Retorno

byte
Retorna el valor del código de estado del desvío:
0 → El desvío se encuentra OK y en servicio.
1 → Existe un error almacenado en el desvío, aunque puede estar en movimiento.
2 → Error de configuración en el desvío.
3 → Otro tipo de error.

Parámetros

1. word WMKAddress
Número de desvío que se desea consultar.

116

```
void WMKStopMovement (word WMKAddress)
```

Descripción general

Detiene el movimiento de una desvío (cuando se estaba moviendo en manual)

Retorno

void
Este método no retorna ningún valor.

Parámetros

1. word WMKAddress
Número de desvío que se desea detener.

117

```
bool WMKTimeoutError (word WMKAddress)
```

Descripción general

Devuelve cierto si se ha producido el desvío no ha alcanzado la posición solicitada en un tiempo determinado

Retorno

true
El desvío no ha alcanzado la posición
false
El desvío ha alcanzado la posición o aún no se ha cumplido el tiempo máximo

Parámetros

1. word WMKAddress
Número de desvío que se desea consultar.

118

```
bool WMKWorking (word WMKAddress)
```

Descripción general

Devuelve cierto si un desvío está realizando un cambio de posición

Retorno

true
El desvío está realizando el cambio
false
El desvío está en una posición fija parado

Parámetros

1. word WMKAddress
Número de desvío que se desea consultar.

119

```
void ZonesSubPaths(word Source,
                   word Target,
                   word &SubPath1,
                   word &SubPath2,
                   word &SubPath3,
                   word &SubPath4,
                   word &SubPath5)
```

Descripción general

Obtiene las posibles trayectorias para ir de un punto a otro (segmentos).

Retorno

void
Este método no retorna ningún valor.

Parámetros

1. word Source
Zona de inicio.
2. word Target
Zona de destino.
3. word SubPath1
Parámetro de salida que indica la trayectoria 1 para ir de la zona de inicio a la zona de destino.
4. word SubPath2
Parámetro de salida que indica la trayectoria 2 para ir de la zona de inicio a la zona de destino.
5. word SubPath3
Parámetro de salida que indica la trayectoria 3 para ir de la zona de inicio a la zona de destino.
6. word SubPath4
Parámetro de salida que indica la trayectoria 4 para ir de la zona de inicio a la

zona de destino.
7. word SubPath5
Parámetro de salida que indica la trayectoria 5 para ir de la zona de inicio a la zona de destino.

4.2.29.4 Utilización del componente

4.2.29.4.1 Topología del sistema

El sistema completo se divide en distintas *zonas*, que se definen durante su configuración. Cada zona está compuesta a su vez de distintos segmentos o puntos de ruta que son la unidad lógica mínima en la que se divide el sistema.

De esta manera, se pueden definir *trayectorias* en la electrovía como una sucesión de segmentos. Estas trayectorias son definidas a priori, incorporadas a la configuración del sistema y se mantendrán inalterables durante todo su funcionamiento.

Las trayectorias definen caminos que un vehículo puede seguir, pero en ningún caso definen un movimiento concreto de un vehículo. El movimiento de un vehículo está definido a partir de una trayectoria, un segmento inicial desde el que parte (que pertenece a la trayectoria) y un segmento final al que llega (que también pertenece a la trayectoria). Estos movimientos no están definidos a priori, y es el programador de control el que decide qué movimiento se puede realizar.

Existe otra clase de movimiento, de más alto nivel, que involucra varias zonas. Este tipo de movimiento se define en la configuración y especifica las posibles trayectorias que un vehículo puede seguir para viajar entre dos zonas.

4.2.29.4.2 Configuración

La configuración del sistema se realiza a través de un archivo llamado *RailBusConfig.ini*, colocado en el directorio de Galileo. En este archivo se especifica por un lado la topología del sistema (DKZs y Zonas) y las posibles trayectorias que pueden seguir los vehículos.

4.2.29.4.2.1 Sección General

En esta sección se especifica una parametrización básica del componente:

- **WMKOffset**: Desplazamiento en la dirección de los WMKs (desvíos). Todos los desvíos tienen direcciones consecutivas, por lo que fijando aquí el offset, se les puede referenciar con direcciones 1,2,3, etc.
- **DelayedWrite**: Modo de envío de rutas
  - 0 → En cada ciclo de programa se comprueba si hay rutas a enviar y se envían (se ordenan por orden de creación del trolley en Galileo)
  - 1 → La ruta se encola en el componente y este las envía por orden de antigüedad

- **DelayTime**: Tiempo de espera necesario antes de enviar un mensaje para permitir al DKZ procesarlo

- **BuffersEnabled**: Habilitación del uso de buffers.
- **AutoBypass**: Habilita el modo "Bypass position" para los carros. En este modo no es necesario gestionar el bit de bypass (lo hace el propio DKZ) para iniciar/parar los carros

Ejemplo

```
[General]
WMKOffset=400
DelayedWrite = 1;
DelayTime = 150
BuffersEnabled = 0
AutoBypass = 1
```

4.2.29.4.2.2 Sección DKZS

En esta sección se especifican los DKZs que están presentes en el sistema, indicando su dirección Profibus. Se compone de varias líneas en cada una de las cuales se indica el DKZ y su dirección.

```
Nombre_DKZ = Dirección Profibus
```

El nombre del DKZ no tiene importancia en el desarrollo del resto del componente, y solo es necesario a modo informativo.

Ejemplo

```
[DKZS]
DKZ1=12
DKZ2=13
```

4.2.29.4.2.3 Sección SEGMENTS

Esta sección permite definir de manera exacta todos y cada uno de los segmentos de la instalación, de manera que se pueda corregir uno de los problemas que tienen el sistema de LJU. Esté sistema, actualiza la información del segmento en el que se encuentra el trolley cada cierto tiempo que depende del número de trolley dentro de la instalación. Esto significa que si se quiere hacer un control basado en segmentos (más sencillo), la información acerca de donde está posicionado el trolley en cada momento puede estar desactualizada, ya que pudo haber cambiado de segmento sin haberse actualizado esta información aún. La solución pasa por utilizar la información física, que se actualiza constantemente. Con los datos de esta sección, el componente puede realizar la conversión automática de posición física a segmento y trabajar con este último en la mayoría de las ocasiones.

En esta sección hay que codificar para cada segmento su posición de inicio y su posición de finalización.

Ejemplo

```
[SEGMENTS]
205=15895,18459
206=18460,20458
```

4.2.29.4.2.4 Sección SWITCHES

En esta sección se indican cuáles de los segmentos definidos pertenecen a desvíos.

La sección **Switches** solo dispone de un campo en el que se indican el número de desvíos a configurar:

```
SWInfoCount=numSwitches
```

Por cada desvío que se defina es necesario añadir una sección [Switch*N*] donde se definirán los datos individuales de cada desvío. El formato de esta sección es el siguiente:

```
[SwitchN]
Address = número de desvío
Segments= Indica que segmentos se quieren tratar de manera especial
por el programa de control. No tienen por qué ser realmente los
segmentos propios del desvío, pueden ser segmentos anteriores o
posteriores
Entrances = número de entradas del desvío
Exits = número de salidas del desvío
```

Ejemplo

```
[SWITCHES]
SWInfoCount=2

[SW1]
Address=1
Segments=242,243,250
Entrances=1
Exits=2

[SW2]
Address=2
Segments=205,252,251
Entrances=2
Exits=1
```

4.2.29.4.2.5 Sección PATHS

En esta sección se especifican las distintas trayectorias o caminos posibles en la electrovía. Cada camino tiene asociado un nombre por razones informativas, aunque en el resto del componente se utilizará el orden en que están definidos para referenciarlos. Es importante tener en cuenta que en todo el componente se utilizan índices de base uno, es decir, el primer camino definido tendrá el índice uno, el segundo el dos y así sucesivamente. El formato de cada línea es el siguiente:

```
NombreCamino = Lista de segmentos
```

Donde Lista de segmentos es una lista separada con comas con los segmentos (puntos de ruta) que componen el camino.

Ejemplo

```
[Paths]
Path1=123,124,125,126,127 ;Index=1
Path2=1,2,3,4,5 ;Index=2
```

4.2.29.4.2.6 Sección Zones

Las zonas son divisiones a nivel lógico de la instalación. Una zona comprende un conjunto del total de segmentos del sistema. Al igual que en el caso anterior, para definir las zonas se utiliza un nombre informativo y una serie de segmentos. Cada zona se referencia con un índice que viene dado por el orden en que se definen en el fichero .ini (índice de base uno). El formato utilizado para la definición de zonas es el siguiente:

```
NombreZona = Lista de segmentos
```

Donde Lista de segmentos es una lista separada con comas con los segmentos (puntos de ruta) que pertenecen a la zona.

Ejemplo

```
[Zones]
Zona1=123,124,125,126,127,128,19,130 ;Index=1
Zona2=1,2,3 ;Index=2
```

4.2.29.4.2.7 Sección Stations

En esta sección se definen las distintas estaciones presentes en la instalación, así como el segmento en que están colocadas y la máquina a la que pertenecen. La sección [Stations] solo dispone de un campo en el que se indican el número de estaciones a configurar:

```
StationsCount=numStations
```

Por cada estación que se defina es necesario añadir una sección [Station*N*] donde se definirán los datos individuales de cada estación. El formato de esta sección es el siguiente:

```
[StationN]
Number=número de estación
Type=tipo de estación
Segments=Lista de segmentos que ocupa (uno por subestación)
Machines= Lista de máquinas (uno por subestación)
Position=Lista con las posiciones en mm de cada subestación
LoadWay=Tipo de carga y descarga; 1 = Carga sentido a izquierdas
;                                 2 = Carga sentido a derechas
```

```
;                                   3 = Descarga sentido a izquierdas
;                                   4 = Descarga sentido a derechas
;                                   5 = Carga & Descarga a izquierdas
;                                   6 = Carga & Descarga a derechas
```

Ejemplo

```
[Stations]
StationsCount=1

[Station1]
Number=5
Type=1
Segments=9,10,11
Machines=2,4,6
Position=1250,1300,1350
LoadWay=1
```

4.2.29.4.2.8 Sección Movements

En esta sección se definen las posibles trayectorias que un vehículo puede seguir para moverse desde una zona a otra. Al igual que en el caso anterior, en cada línea se definen todos los parámetros del movimiento:

```
NombreMovimiento = ZonaInicio,ZonaFin,Lista de trayectorias
```

Donde:

- *NombreMovimiento* nombre informativo del movimiento
- *ZonaInicio*: zona en la que se inicia del movimiento. Índice de base 1 que referencia una zona definida en la sección Zones.
- *ZonaFin*: zona en la que se finaliza el movimiento. Índice de base 1 que referencia una zona definida en la sección Zones.
- Lista de trayectorias: lista separada por comas de los paths que se utilizarán al generar la ruta

Ejemplo

```
[Movements]
Mov1= 1,2,1,3,2
```

Es responsabilidad del programador de control interpretar los índices de las trayectorias que se introducen en la configuración. De esta manera el ejemplo anterior se puede interpretar como que para ir de la zona uno a la dos solo se deben utilizar las trayectorias 1, 2 y 3

4.2.29.4.2.9 Sección AutoMovements

Esta sección es similar a la anterior, pero referida a los movimientos que seguirá el carro cuando se le mande "**apartarse**" con una orden *MoveTrolley*. A diferencia de la anterior, no necesita zona final, ya que estos movimientos solo tienen en cuenta la zona inicial.

```
NombreMovimiento = ZonaInicio, Lista de trayectorias
```

Donde:

- N*ombreMovimiento* nombre informativo del movimiento
- *ZonaInicio*: zona en la que se inicia del movimiento. Índice de base 1 que referencia una zona definida en la sección Zones.
- *Lista de trayectorias*: lista separada por comas de los paths que se utilizaran al generar la ruta

Ejemplo

```
[AutoMovements]
AMV1=1,1,2,4,1
AMV2=2,2,4,1,2
AMV3=3,3,4,3
AMV4=4,4,1,2,4
```

4.2.29.4.2.10 Sección ForbiddenSegments

Esta sección indica segmentos anteriores a desvíos con más de una posible salida. Estos segmentos tendrán una gestión especial a la hora de seleccionar un carro el RBM.

```
Segments = lista de segmentos
```

Ejemplo

```
[ForbiddenSegments]
Segments = 241,212,259,308,309
```

4.2.29.4.2.11 Sección StopSegments

Esta sección indica segmento/s de parada intermedia con el fin de evitar que un carro reciba una nueva orden p.ej en el segmento 240 y por retardo en las comunicaciones con los DKZ se reciba en uno posterior (241, 242). Si esta nueva orden obliga al carro a salir por el tramo "no adecuado" (diferente al seleccionado en movimientos de "move away" y de "buffer" que son gestionados internamente por GALILEO. Los generados por control no se han de filtrar en este punto (se deben filtrar por control).

```
Segments = lista de segmentos
```

Ejemplo

```
[StopSegments]
Segments = 241,
```

4.2.29.4.2.12 Sección ForbiddenBufferSegments

Esta sección Define segmentos prohibidos para carros que se mueven a buffer.

```
Segments = lista de segmentos
```

Ejemplo

```
[ForbiddenSegments]
Segments = 212,259,242,259
```

4.2.29.4.2.13 Sección ForbiddenLoads

Esta sección define los transportadores orígenes en los que NO se puede realizar una carga si el carro ya tiene un tracking y va a realizar la descarga en un transportador dado.

```
TransportadorDestinoDescarga = lista de transportadores prohibidos
para realizar posteriormente una carga
```

Ejemplo

```
[ForbiddenLoads]
; Por ejemplo (ESTACION 52) Descarga "automática" hacia Mezclas Palet
vacío -> Se permite carga desde TR369 y TR370
366=377,433,397,438,361,
```

4.2.29.4.2.14 Sección Buffers

Esta sección define los buffers de la electrovía. Un buffer es un segmento al cual puede ser enviado un carro, que previamente ha finalizado otra orden, tras recibir una orden de posicionarse en él. Varios carros pueden ser enviados al mismo buffer, éstos se encolan desde el segmento del buffer.

```
NombreBuffer = número segmento, posición en mm
```

Ejemplo

```
[Buffers]
b1=231,73944
b2=234,87912
```

4.2.29.4.3 Inicialización

Puesto que existen dos componentes que es necesario utilizar para controlar la electrovía es necesario tener ciertas cosas en cuenta a la hora de inicializar el sistema:

- Se crearán tantos elementos de bus de como DKZ existan en la instalación. Estos elementos estarán asociados a los componentes DKZ, y ocuparan 228 `bytes` de entrada y 130 `bytes` de salida.
- Se creará un único elemento de bús que estará asociado al componente Railbus. No ocupará espacio de E/S.
- El método *`Init()`* del componente RailBus inicia el control de ésta. Este componente es que coordina la correcta comunicación con los dispositivos (DKZs), por lo que es NECESARIO que antes de que se inicie la ejecución se hayan añadido al componente RailBus las referencias a los DKZ presentes en el sistema mediante el método `AddDKZ`. Una manera de hacer esto podría ser la siguiente:
  1. Se crea una variable global de tipo RailBus asociada al elemento de bus correspondiente (sin espacio de E/S).
  2. Se crea un variable global de tipo `word` que sirva de contador de DKZ inicializados.
  3. En el arranque del sistema se inicializará este contador a 0
  4. Se crea un secuenciador por cada DKZ. Este secuenciador simplemente iniciará el DKZ. Para ello, cada uno contendrá un parámetro de tipo DKZComp. En el arranque de este secuenciador, se obtendrá el ID mediante el método `GetMyID()` del componente DKZComp. Después, se añadirá el DKZ a la electrovía llamando al método `AddDKZ`, indicando el ID obtenido y la dirección Profibus. Por último, se incrementará el contador de DKZs.
  5. No se deberá realizar la llamada al método `Init`() del componente RailBus, hasta que el contador de DKZ no alcance el número de DKZ que tengamos en la instalación.
  6. A partir de este momento, ya se pueden realizar las operaciones deseadas con la electrovía. Todas estas operaciones se realizarán mediante el componente RailBus (accesible por todos), no volviendo a utilizar el componente DKZComp.

4.2.29.4.4 Creación y ejecución de caminos

Una de las partes más importantes a la hora de configurar el sistema es la creación correcta de las rutas. Las rutas se crean mediante una sucesión de segmentos (tabla de ruta) que serán enviados al carro para que éste realice el movimiento. El carro dependiendo de cómo sean estos segmentos puede estar en los siguientes estados:

- **La tabla de ruta está vacía**. Esto se produce por ejemplo, cuando termina de ejecutar una tabla de ruta y no se le manda una nueva. En este caso, el display del carro muestra el error ***-86***.
- **Hay tabla de rutas pero incorrecta**. El carro ha recibido una tabla de rutas, pero le resulta imposible ejecutarla. El error más común en estos casos es que el primer segmento de la nueva tabla de rutas no sea accesible desde la posición actual del carro. Por ejemplo, el carro está

situado en el segmento 215, cuyos segmentos contiguos son el 214 y el 216 (izquierda y derecha). Si se le manda una tabla cuyo primer segmento sea el 217, se producirá este error. El carro se parará en el segmento 215 y mostrará en el display el número de segmentos cargados en la nueva tabla (realmente no se trata de un error en la tabla, sino de un error en la posición del carro). Para solucionar este error se pueden realizar tres acciones:

- Enviar una nueva tabla de rutas cuyo primer punto sea accesible
- Mediante el control remoto, mover el carro al primer segmento de la ruta cargada
- Mediante la consola de programación MU, modificar la tabla cargada para que el primer punto sea accesible.

Es importante tener en cuenta que un carro iniciará su movimiento cuando el primer punto de la ruta sea el mismo en el que está o uno de sus contiguos.

- **La ruta es correcta**. En este caso el carro se moverá por la ruta y mostrará en el display el número de puntos de la ruta cargada.

La ejecución de un camino por parte de un carro se realiza de manera transparente para el programa de control. El componte se encarga de manera automática de generar las rutas necesarias para llegar al destino. No obstante el programa de control puede querer cargarle una nueva tabla de ruta (si la necesita) antes de que el carro termine la actual, de manera que no se pare. Para ello, es necesario controlar el segmento en el que está el carro en todo momento. Se pueden utilizar básicamente dos estrategias para saber en qué momento se va a terminar la ejecución de una tabla de rutas:

- Controlando en todo momento el segmento en el que está el carro mediante el método `GetTrolleyRoutePoint`. Con esta estrategia, el programa de control compararía la posición actual con la del segmento en el que tengamos previsto enviar la nueva tabla.
- Mediante el método `GetTrolleyStatusCode`. Este es el método más adecuado (y el recomendado por LJU). Este método devuelve un código indicando el estado del carro. Los valores posibles son los siguientes:
  - 0 → OK
  - 1 → Moviéndose, tabla de ruta finaliza en el próximo segmento
  - 2 → Parado debido a errores en la ruta
  - 3 → Parado debido a otro error

  Cuando se detecte el valor (1), el componente debe preparar una nueva ruta y enviarla. De este modo el carro no se parará. Puesto que al enviar una nueva tabla se elimina la que se está ejecutando en el carro, es necesario tener en cuenta lo siguiente:

  1. El punto en el que está se termina de ejecutar. Aunque se escriba una nueva tabla de rutas, el carro finalizará el movimiento para el punto en que está.

2. Puesto que se envía la nueva tabla cuando el carro está en el *penúltimo punto*, es necesario que la siguiente tabla incluya también el último punto de la ruta anterior.

4.2.29.4.4.1 Errores comunes en las tablas de rutas

- **Se envía una orden al carro y éste no se mueve**. Existen varios errores que explican este problema. Los más frecuentes son:
  - **El DKZ está en modo manual**. En este modo, se ignoran todos los mensajes con tablas de ruta que llegan. La solución pasa por poner el DKZ en modo automático (desde Galileo o desde la consola del propio DKZ).
  - **La tabla enviada tiene el primer punto inaccesible**. Es necesario enviar una nueva tabla de rutas, modificar la existente o mover el carro con el control remoto hasta el punto inicial.
  - **El carro está en modo manual**. Por ejemplo, cuando se utiliza el control remoto se pasa el carro a manual, por lo que no ejecutará la tabla de ruta hasta que no se pase de nuevo a automático.
- **El carro inicia el movimiento, pero en un punto de la ruta se para**. El error más común en este caso es que se envió una nueva tabla de ruta antes de tiempo o la ruta que se está ejecutando está mal definida (la sucesión de segmentos no correcta y hay segmentos no accesibles desde el anterior).
- El carro inicia el movimiento, pero cuando cambia de segmento, cambia el sentido y vuelve al segmento inicial y así sucesivamente. Este error se produce cuando por error, **se envía la misma tabla de ruta continuamente**. Al ser siempre la misma, mientras está en el primer segmento el carro se mueve sin problemas. Al cambiar al siguiente segmento y recibir de nuevo la misma tabla ruta, el primer segmento que se encuentra es precisamente el que acaba de abandonar, pero que es accesible, por lo que cambia el sentido y se va hacia él. La solución consiste en enviar la tabla de ruta una única vez.

4.3 Librería de componentes de comunicaciones (COMCOMPONENTS.dll)

El Sistema de Control Galileo no es un sistema cerrado completamente, sino que permite un cierto grado de comunicación con elementos externos a él, como por ejemplo, Sistemas de Bases de Datos. Dicha comunicación se realiza a través de los componentes que se encuentran en esta biblioteca.

La característica primordial de los componentes de esta librería es que las comunicaciones que utilizan son realizadas de forma asíncrona con respecto a la ejecución de Galileo, es decir, el Sistema de Control Galileo no puede detenerse a esperar por la comunicación de uno de estos componentes, sino que han de proporcionar un mecanismo sencillo de seguir el estado de una comunicación sin

que el Sistema de Control Galileo deba detener sus funciones normales por ello. Debido a esto, estos componentes deberían ejecutarse en un hilo independiente.

4.3.1 ORACLE

4.3.1.1 Interface

El propósito de este apartado es explicar como se consigue comunicar el Sistema de Control Galileo con una Base de Datos Oracle, a través de la invocación de procedimientos PL/SQL almacenados en dicha Base de datos.

Este componente expone el siguiente interface:

Class	Oracle
BUS IN	0 bytes
BUS OUT	0 bytes

Listado de métodos:

1	dword GetContext(string Procedure, dword InputParamCount, dword OutputParamCount, bool &TriggerValue)
2	dword GetParameter(dword Context, dword Index)
3	bool HasNewData(dword Context)
4	void SetConnection(string User, string Password, string Connection)
5	void SetParameter(dword Context, dword Index, dword Value)

Descripción de métodos:

1

```
void SetConnection(string User,
                   string Password,
                   string Connection)
```

Descripción general

Este método debe invocarse en las rutinas de arranque, en él se establecen los parámetros básicos de cualquier comunicación con una BD Oracle.

Retorno

void

Este método no retorna ningún valor.

Parámetros

1. `string` `User`
   Contiene el nombre de un usuario con los permisos necesarios para conectar a una BD Oracle y ejecutar procedimientos almacenados en ella.
2. `string` `Password`
   Debe de contener la palabra de paso o clave del usuario antes mencionado para entrar en una BD Oracle.
3. `string` `Connection`
   Debe de encontrarse la cadena de conexión a una BD Oracle que queramos usar.

2

```
dword GetContext(string Procedure,
                 dword InputParamCount,
                 dword OutputParamCount,
                 bool &TriggerValue)
```

Descripción general

Este método sirve para localizar un contexto desde el que se lanzará la llamada a un procedimiento almacenado en Oracle.
Dicho número de contexto es la manera de identificar cada llamada desde Xana y de obtener información sobre el estado de una consulta. Este método suele ser invocado en las rutinas de arranque.

Retorno

`dword`
Devuelve un identificador de contexto en ejecución.

Parámetros

1. `string` `Procedure`
   Nombre del procedimiento PL/SQL almacenado en la base de datos..
2. `dword` `InputParamCount`
   Número de parámetros de entrada del procedimiento PL/SQL.
3. `dword` `OutputParamCount`
   Número de parámetros de salida del procedimiento PL/SQL.
4. `bool` `&TriggerValue`
   Esta variable pasada por referencia será el disparador que nos permita gobernar cuando se lanza la ejecución del procedimiento almacenado. En el momento en que se invoca este método, la variable se registra, y cuando, desde cualquier parte de nuestro código establezcamos esta variable al valor `true`, comenzará a ejecutarse dicho procedimiento. Una vez que la ejecución del mismo termine, dicha variable pasará automáticamente al valor `false`. Es por ello importante asegurar que la variable de disparo tenga un valor false cuando se invoque este método, a no ser que queramos desencadenar inmediatamente la llamada al procedimiento remoto.

3

```
void SetParameter(dword Context,
                  dword Index,
                  dword Value)
```

Descripción general

Método que se utilizará para establecer el valor de un parámetro del procedimiento

PL/SQL almacenado en BD que se está ejecutando.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `dword Context`
   Número de contexto en el que se busca parámetro.
2. `dword Index`
   Índice del parámetro requerido.
3. `dword Value`
   Valor al que se desea establecer el parámetro.

4

```
dword GetParameter(dword Context,
                   dword Index)
```

Descripción general

Método que se utilizará para averiguar el valor de un parámetro del procedimiento PL/SQL almacenado en BD que se está ejecutando.

Retorno

`dword`

Retorna el valor del parámetro que se solicita.

Parámetros

1. `dword Context`
   Número de contexto en el que se busca parámetro.
2. `dword Index`
   Índice del parámetro requerido.

5

```
bool HasNewData(dword Context)
```

Descripción general

Método que se utilizará para saber si un procedimiento invocado previamente ha finalizado sin error y obtenido algún valor nuevo. La invocación de este método es análoga a comprobar el valor del trigger.

Retorno

`true`

Si el procedimiento ha obtenido datos nuevos.

`false`

Si el procedimiento aún no se ha actualizado.

Parámetros

1. `dword Context`
   Número de contexto en el que se busca parámetro. requerido.

4.3.2 SERVERSOCKET

4.3.2.1 Interface

El propósito de este apartado es explicar la utilización de este componente que realiza las funciones de servidor de sockets.

Servidor de sockets es aquella aplicación que permite la conexión de aplicaciones ajenas y que tiene la capacidad de responder a sus peticiones. La propia arquitectura de los servidores de sockets hace que no puedan enviar datos a los clientes nada más que bajo petición de estos.

Este componente expone el siguiente interface

Class	ServerSocket
BUS IN	0 bytes
BUS OUT	0 bytes

Listado de métodos:

1	void ClearOutputBuffer()
2	void Configure(dword Port, dword DimInBuffer, dword DimOutBuffer)
3	byte GetInputByte(dword Index)
4	byte GetOutputByte(dword Index)
5	void Open()
6	void SetInputByte(dword Index, byte Value)
7	void SendOutputBuffer()
8	void SetOutputByte(dword Index, byte Value)

Descripción de métodos:

1

```
void Configure(dword Port,
               dword DimInBuffer,
               dword DimOutBuffer)
```

Descripción general

Este método debe invocarse en las rutinas de arranque, en él se establecen los parametros básicos de cualquier comunicación.

Retorno

void

Este método no retorna ningún valor.

Parámetros

1. dword Port

   Indica el puerto de sockets que se abrirá por este servidor de sockets. Ninguna otra aplicación puede estar utilizando este puerto de sockets o fallará la

apertura. No es necesario indicar que direcciones IP hay que manejar ya que el sistema abrira dicho puerto sobre todos los interfaces conectados al sistema (Tanto tarjetas de red como conexiones RAS establecidas).

2. `dword DimInBuffer`

Dimensión del buffer de entrada que queremos establecer. En este buffer se recibirán los datos enviados por el cliente.

3. `dword DimOutBuffer`

Dimensión del buffer de salida en el cual escribiremos los datos que queremos transmitir al cliente.

2

```
void Open()
```

Descripción general

Este método abre efectivamente la comunicación con los datos establecidos mediante el método `Configure`. Debe ser invocado por lo tanto despues de haber invocado `Configure`. Este método suele ser invocado en las rutinas de arranque.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

3

```
byte GetInputByte(dword Index)
```

Descripción general

Este método nos permite consultar el valor de los datos escritos por el cliente en el buffer de entrada.

Retorno

`byte`

Retorna el valor del byte solicitado.

Parámetros

1. `dword Index`

Número del byte que se desea leer. Debe de estar entre 0 y el tamaño del buffer configurado menos 1.

4

```
void SetInputByte(dword Index,
                  byte Value)
```

Descripción general

Este método nos permite escribir sobre el buffer de entrada. Su única función es alterar o eliminar los datos escritos por el cliente.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `dword Index`
   Número del byte que se desea leer. Debe de estar entre 0 y el tamaño del buffer configurado menos 1.
2. `byte Value`
   Valor que se desea asignar al byte indicado.

5

```
byte GetOutputByte(dword Index)
```

Descripción general

Este método nos permite consultar el valor de los datos en el buffer de salida.

Retorno

`byte`
Retorna el valor del byte solicitado.

Parámetros

1. `dword Index`
   Número del byte que se desea leer. Debe de estar entre 0 y el tamaño del buffer configurado menos 1.

6

```
void SetOutputByte(dword Index,
                   byte Value)
```

Descripción general

Este método nos permite escribir sobre el buffer de salida. Estos datos son los que se enviarán al cliente.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `dword Index`
   Número del byte que se desea leer. Debe de estar entre 0 y el tamaño del buffer configurado menos 1.
2. `byte Value`
   Valor que se desea asignar al byte indicado.

7

```
void ClearOutputBuffer()
```

Descripción general

Este método borra el contenido del buffer de Salida. Suele ser utilizado antes de volver a escribir una respuesta.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

8
`void SendOutputBuffer()`
Descripción general
Este método envía al cliente los datos que se encuentran en el buffer de salida.
Retorno
`void` Este método no retorna ningún valor.
Parámetros
Este método no requiere parámetros de entrada.

NOTA: *El cliente de sockets debe terminar el envío con un retorno de carro y un cambio de línea. El servidor de sockets añadirá al final del telegrama escrito por control los correspondientes retornos de carro y cambio de línea.*

4.3.3 SERVERSOCKETRAW

4.3.3.1 Introducción

El propósito de este apartado es explicar la utilización de este componente que realiza las funciones de servidor de sockets genérico.

A diferencia del componente ServerSocket, éste permite la creación de sockets tanto cliente como servidores y el formato de los datos a enviar y recibir es libre, sin necesidad de que finalicen con un salto de línea.

4.3.3.2 Funcionamiento básico

Es necesario entender el funcionamiento básico del componente y la manera en la que se realizan las comunicaciones para poder utilizarlo correctamente.

El componente permite crear conexiones con otros computadores a nivel de sockets TCP, en los que este socket puede funcionar en el papel de cliente o en el papel de servidor. En ambos casos, el funcionamiento básico es similar, variando únicamente la configuración inicial del componente.

4.3.3.3 Configuración

La configuración del componente se realiza mediante las funciones ConfigAsServer o ConfigAsClient. Cada una de estas funciones permite configurar el socket como servidor o como cliente. Estas funciones se invocarán normalmente en la etapa de arranque del secuenciador.
Es importante tener en cuenta que estas funciones no establecen ninguna conexión, simplemente la preparan, pero es necesario invocar la función Open para establecer

esta conexión. Esta función intenta realizar la conexión, aunque su funcionamiento es distinto dependiendo de si se trata de un cliente o de un servidor:

- Cliente: intenta conectarse al host remoto de manera activa. Esta función se debería invocar una única vez, ya que es el componente el que internamente mantendrá la conexión y las realizará las posibles reconexiones en caso de perdidas de comunicación.
- Servidor: inicia la escucha de conexiones. Es decir, en el caso de un servidor, no se realiza una conexión, ya que esto es función del host remoto. Lo que hace la función Open es iniciar la escucha de conexiones. Al igual que en el caso de los sockets cliente, esta función debe invocarse una única vez, ya que el componente se encarga de gestionar la conexión.

4.3.3.4 Envío de datos

El envío de datos se realiza a través de un buffer cuyo tamaño se decide durante la configuración. Las funciones implicadas en el envío de datos son:

- *GetOutputByte*: obtiene un byte almacenado en el buffer de salida.
- *SetOutputByte*: carga un byte en el buffer de salida.
- *ClearOutputBuffer*: pone a cero el buffer de salida
- *SendOutputBuffer*: inicia el envío por red de los datos del buffer de salida
- *WriteFinished*: indica cuando el envío esta completado

En el modo de funcionamiento normal, el usuario lo único que tendrá que hacer para realizar un envío es seguir el siguiente esquema:

1. Esperar a que no haya otro envío en curso (con el método `WriteFinished`)
2. Vaciar el buffer de salida y cargar los datos (métodos `ClearOutputBuffer` y `SetOutputByte`)
3. Enviar los datos (método `SendOutputBuffer`)
4. Esperar a que termine el envío (método `WriteFinished`).

4.3.3.5 Recepción de datos

La recepción de datos se realiza mediante un sistema de doble buffer, de manera que el componente pueda seguir recibiendo datos incluso cuando el usuario esta tratando aun datos recibidos anteriormente. Este sistema es en gran parte transparente para el usuario. El funcionamiento de la recepción de datos es el siguiente:

1. El componente recibe los datos y los almacena en un buffer interno. Este buffer interno no es accesible desde el programa de control.

2. A petición del usuario, los datos de este buffer interno se copian al buffer de usuario (buffer en entrada), que tiene el mismo tamaño del buffer interno. Este buffer ya es accesible por el usuario.

Las funciones implicadas en la recepción de datos son:

- *GetInputByte*: obtiene un dato del buffer de entrada (buffer de usuario)
- *SetOutputByte*: almacena un dato en el buffer de entrada (buffer de usuario)
- *ClearInputBuffer*: pone a cero el buffer de entrada (buffer de usuario)
- *SetFinishChar*: configura el componente para que utiliza un carácter determinado de terminador de lectura
- *isFinishRead*: comprueba en el buffer interno (NO EN EL DE USUARIO) si se encuentra el carácter terminador configurado.
- *GetInputBuffer*: realiza la copia del buffer interno al de usuario

4.3.3.6 Interface

Este componente expone el siguiente interface

Class	ServerSocketRaw
BUS IN	0 bytes
BUS OUT	0 bytes

Listado de métodos:

1	void ConfigureAsServer(dword Port, dword DimInBuffer, dword DimOutBuffer, bool Delay)
2	void ConfigureAsClient(string Host, dword Port, dword DimInBuffer, dword DimOutBuffer, bool Delay)
3	void Open()
4	byte GetInputByte(dword Index)
5	void SetInputByte(dword Index, byte Value)
6	byte GetOutputByte(dword Index)
7	void SetOutputByte(dword Index, byte Value)
8	void ClearOutputBuffer()
9	void SendOutputBuffer()
10	void SetFinishChar(word Character)
11	bool isFinishRead()
12	bool WriteFinished()

13	`void GetInputBuffer()`
14	`void Close()`
15	`void SetOutputNumber(word Index, word Length, word Value)`
16	`word GetInputNumber(word Index, word Length)`
17	`void SetFinishLen(dword Length)`
18	`void ClearInputBuffer()`
19	`dword GetInputBufferLen()`

Descripción de métodos:

1

```
void ConfigureAsServer(dword Port,
                       dword DimInBuffer,
                       dword DimOutBuffer,
                       bool Delay)
```

Descripción general

Este método debe invocarse en las rutinas de arranque y es el que configura el componente para trabajar como un servidor.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `dword Port`
   Indica el puerto de sockets que se abrirá por este servidor de sockets. Ninguna otra aplicación puede estar utilizando este puerto de sockets o fallará la apertura. No es necesario indicar que direcciones IP hay que manejar ya que el sistema abrira dicho puerto sobre todos los interfaces conectados al sistema (Tanto tarjetas de red como conexiones RAS establecidas).
2. `dword DimInBuffer`
   Dimensión del buffer de entrada que queremos establecer. En este buffer se recibirán los datos enviados por el cliente.
3. `dword DimOutBuffer`
   Dimensión del buffer de salida en el cual escribiremos los datos que queremos transmitir al cliente.
4. `bool Delay`
   Con este parámetro en el valor `true` se activa el alogaritmo *Nagle* que retrasa las comunicaciones para maximizar el tamaño de los envíos. Si su valor se coloca a `false` los datos son enviados inmediatamente sin ningún tipo de retardo. Para un funcionamiento normal colocar este parámetro a `true`.

2

```
void ConfigureAsClient(string Host,
```

```
                        dword Port,
                        dword DimInBuffer,
                        dword DimOutBuffer,
                        bool Delay)
```

Descripción general

Este método debe invocarse en las rutinas de arranque y es el que configura el componente para trabajar como un cliente.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `string Host`
   Cadena con el nombre del Host al que se desea conectar.
2. `dword Port`
   Indica el puerto de sockets que se abrirá por este servidor de sockets. Ninguna otra aplicación puede estar utilizando este puerto de sockets o fallará la apertura. No es necesario indicar que direcciones IP hay que manejar ya que el sistema abrira dicho puerto sobre todos los interfaces conectados al sistema (Tanto tarjetas de red como conexiones RAS establecidas).
3. `dword DimInBuffer`
   Dimensión del buffer de entrada que queremos establecer. En este buffer se recibirán los datos enviados por el cliente.
4. `dword DimOutBuffer`
   Dimensión del buffer de salida en el cual escribiremos los datos que queremos transmitir al cliente.
5. `bool Delay`
   Con este parámetro en el valor `true` se activa el alogaritmo *Nagle* que retrasa las comunicaciones para maximizar el tamaño de los envíos. Si su valor se coloca a `false` los datos son enviados inmediatamente sin ningún tipo de retardo. Para un funcionamiento normal colocar este parámetro a `true`.

3

```
void Open()
```

Descripción general

Este método abre efectivamente la comunicación con los datos establecidos mediante alguno de los métodos `Configure`. Debe ser invocado por lo tanto despues de haber invocado `Configure`. Este método suele ser invocado en las rutinas de arranque.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

4

```
byte GetInputByte(dword Index)
```

Descripción general

Este método nos permite consultar el valor de los datos escritos por el cliente en el buffer de entrada.

Retorno

`byte`

Retorna el valor del byte solicitado.

Parámetros

1. `dword Index`
   Número del byte a leer, debe de estar entre 0 y el tamaño del buffer configurado menos 1.

5

```
void SetInputByte(dword Index,
                  byte Value)
```

Descripción general

Este método nos permite escribir sobre el buffer de entrada. Su única función es alterar o eliminar los datos escritos por el cliente.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `dword Index`
   Número del byte a leer, debe de estar entre 0 y el tamaño del buffer configurado menos 1.
2. `byte Value`
   Valor que se desea escribir en el byte indicado.

6

```
byte GetOutputByte(dword Index)
```

Descripción general

Este método nos permite consultar el valor de los datos en el buffer de salida.

Retorno

`byte`

Retorna el valor del byte solicitado.

Parámetros

1. `dword Index`
   Número del byte que se desea leer. Debe de estar entre 0 y el tamaño del buffer configurado menos 1.

7

```
void SetOutputByte(dword Index,
                   byte Value)
```

Descripción general

Este método nos permite escribir sobre el buffer de salida. Estos datos son los que se enviarán al cliente.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `dword Index`
   Número del byte que se desea leer. Debe de estar entre 0 y el tamaño del buffer configurado menos 1.
2. `byte Value`
   Valor que se desea asignar al byte indicado.

8

```
void ClearOutputBuffer()
```

Descripción general

Este método borra el contenido del buffer de Salida. Suele ser utilizado antes de volver a escribir una respuesta.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

9

```
void SendOutputBuffer()
```

Descripción general

Este método envía al cliente los datos que se encuentran en el buffer de salida.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros de entrada.

10

```
void SetFinishChar(word Character)
```

Descripción general

Este método establece el byte que se utilizará como terminador de lectura.

Retorno

`void`
Este método no retorna ningún valor.

Parámetros

1. `word` `Character`
   Valor comprendido entre 0 y 255 que indica el byte que se utilizará como terminador de la lectura.

11

```
bool isFinishRead()
```

Descripción general

Comprueba si en el buffer interno está el carácter terminador.

Retorno

`true`
El carácter terminador ha sido leído.
`false`
El carácter terminador NO ha sido leído.

Parámetros

Este método no requiere parámetros de entrada.

12

```
bool WriteFinished()
```

Descripción general

Comprueba si no hay operación de escritura en curso.

Retorno

`true`
El componente no tiene nada que escribir.
`false`
El componente tiene datos para escribir.

Parámetros

Este método no requiere parámetros de entrada.

13

```
void GetInputBuffer()
```

Descripción general

Realiza la copia del buffer interno al de usuario.

Retorno

`void`
Este método no retorna ningún valor

Parámetros

Este método no requiere parámetros de entrada.

14

```
void Close()
```

Descripción general

Cierra la conexión establecida.

Retorno

`void`

Este método no retorna ningún valor

Parámetros

Este método no requiere parámetros de entrada.

15

```
void SetOutputNumber(word Index,
                     word Length,
                     word Value)
```

Descripción general

Almacena en la trama de salida, a partir de la posición indicada en el parámetro `Index` y ocupando un longitud de `Length` bytes, la representación en formato cadena del `Value`.

Retorno

`void`

Este método no retorna ningún valor

Parámetros

1. `word Index`
   Posición en la que se comienza la escritura-
2. `word Length`
   Número de bytes que se va a ocupar.
3. `word Value`
   Valor a almacenar.

16

```
word GetInputNumber(word Index,
                    word Length)
```

Descripción general

Evalúa el valor numérico de la cadena que se encuentra a partir del byte indicado en el parámetro `Index` y ocupa un longitud de `Length` bytes en la trama de entrada.

Retorno

`word`

Retorna el valor numérico de la cadena

Parámetros

1. `word Index`
   Posición en la que se comienza la lectura.
2. `word Length`
   Número de bytes que ocupa la cadena.

17

```
void SetFinishLen(dword Length)
```

Descripción general

Este método establece la longitud de trama que se utilizara para detectar el final de una lectura.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

1. `dword Lenght`
   Longitud de la trama que se desea leer.

18

```
void ClearInputBuffer()
```

Descripción general

Borrar los datos del buffer de entrada. Este método borra los datos del buffer "copia", es decir, de los datos obtenidos mediante GetInputBuffer. En caso de desear borrar todos los datos de entrada, incluidos los almacenados en el buffer interno, es necesario llamar en primer lugar a GetInputBuffer, para que realice la copia y borre el buffer interno, y después a ClearInputBuffer, para que borre la copia.

Retorno

`void`

Este método no retorna ningún valor.

Parámetros

Este método no requiere parámetros.

19

```
DWord GetInputBufferLen()
```

Descripción general

Devuelve el tamaño del buffer de datos obtenido mediante GetInputBuffer.

Retorno

`void`

Retorna el número de bytes del buffer.

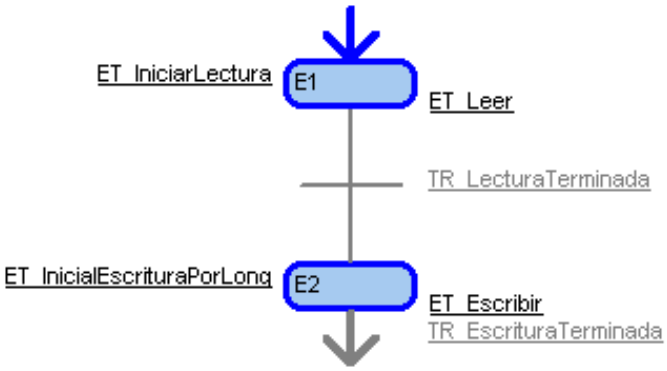
Parámetros

Este método no requiere parámetros.

4.3.3.7 Ejemplo de utilización

A continuación se muestra un ejemplo de utilización de este componente. El ejemplo implementa un socket cliente, que se utilizará para conectar Galileo con un servidor y responder cualquier mensaje que éste envíe con la misma trama que recibe.

El programa cuenta con único sencuenciador que implementa las siguientes etapas:



La implementación de estas etapas es la siguiente:

ET_IniciarLectura

Inicia el proceso de lectura de una trama enviada por el servidor. Lo único que hace es limpiar el buffer de entrada y poner la marca de lectura terminada a false.

```
this.p_LecturaTerminada=false;
this.p_C_Socket->ClearInputBuffer();
```

ET_Leer

Realiza el proceso de lectura. Puesto que la lectura se realiza de manera transparente para el usuario lo único que se hace aquí es comprobar que se termino la lectura. La función isFinishRead devolverá true si se ha recibido ya el carácter terminador o se ha alcanzado la longitud de trama configurada (con SetFinishChar o SetFinishLen)

```
dword i;
if (!this.p_C_Socket->isFinishRead()) return;
// Se ha encontrado el separador
this.p_LecturaTerminada=true;
```

TR_LecturaTerminada

La transición se cumple cuando la marca se active (en la etapa anterior)

```
return this.p_LecturaTerminada;
```

ET_IniciarEscrituraPorLong

Copia la trama recibida al buffer de salida para responder al servidor

```
word i;
word aChar;
this.p_EscrituraTerminada=false;

// Obtnemos el buffer leido, para poder enviarselo al servidor de
nuevo
this.p_C_Socket->GetInputBuffer();
i=0;
while (i<5) // La longitud configurada
{
  aChar=this.p_C_Socket->GetInputByte(i);
  this.p_C_Socket->SetOutputByte(i,aChar);
  i=i+1;
}
this.p_C_Socket->SendOutputBuffer(i);
```

ET_Escribir

Como acción de etapa solo se comprueba que la escritura haya finalizado

```
this.p_EscrituraTerminada=this.p_C_Socket->WriteFinished();
```

TR_EscrituraTerminada

Solo comprueba la marca que indica si la escritura termino

```
return this.p_EscrituraTerminada;
```

Además es necesario un código en arranque del secuenciador que configure el componente:

ET_Configurar

Configura el socket como cliente (con `ConfigureAsClient`) y lo conecta a la IP deseada, puerto 2000, y un tamaño de buffer de entrada y de salida de 200 bytes cada uno.
Ademas configura un tamaño de trama de 5 bytes (`SetFinishLen`), de manera que el método `isReadFinish` devolverá true al alcanzar este tamaño.
Por ultimo conecta el socket con `Open`.

```
// Configuramos el socket como un cliente
//  * IP Remota: "127.0.0.1"
//  * Puerto remoto: 2000
//  * Tamaño del buffer de entrada: 200
//  * Tamaño del buffer de salida: 200
//  * Nagle activado

this.p_C_Socket->ConfigureAsClient(this.p_S_IP,2000,200,200,true);

//Si se quiere controlar la lectura por caracter, descomentar esta
//linea,
//comentar la siguiente, y cambiar la etapa de IniciarEscritura
//this.p_C_Socket->SetFinishChar('@');
```

```
this.p_C_Socket->SetFinishLen(5);
this.p_C_Socket->Open();
```

5 MECALUX TRANSPORT AGENT

El Agente de Transportes es el nombre que reciben los hilos de ejecución del Sistema de Control Galileo que se ocupan del control de búsqueda de órdenes, fines de orden, notificaciones de eventos, etc.

Este servicio sustituye a la aplicación *Monitor de Transportes*. Mientras que la aplicación *Monitor de Transportes* tenia una ejecución monolítica (es decir, todo el código de ejecución reside en el propio ejecutable), para el diseño del Agente de Transportes se ha empleado otra filosofía, la mayoría (99%) del código "inteligente" del Agente de Transportes se encuentra en Base de Datos en un Paquete de PLSQL encriptado de manera que se evite el hecho de que el mismo sea copiado y alterado. Este paquete se denomina *Path Finder* y es instalado desde el Software *Designer*. Mediante este Software se pueden crear las tablas que necesita *Path finder*, instalar los paquetes propiamente dichos, realizar la configuración y transferirla a la base de datos. Aunque no es imposible realizar la instalación y configuración de forma manual, es un procedimiento que no se recomienda, ya que las comprobaciones que se realizan en la configuración desde *Designer* no se pueden aplicar.

La aplicación se ejecuta de forma continua y su principal misión es funcionar como interlocutor de la tarjeta de control para gestionar las diferentes variantes de eventos que se producen en nuestras instalaciones.

A continuación pasaremos a describir a fondo cada una de las particularidades de esta aplicación.

5.1 Instalación

La instalación se efectúa desde *Designer* y en el manual de usuario de *Designer* se explica perfectamente como realizarla.

La configuración del agente de transportes en ningún caso debe de realizarse de manera manual. La aplicación *Designer* sobrescribirá todas las configuraciones realizadas de forma manual.

5.2 Conceptos generales

Mediante el paquete *PathFinder* se pretenden solventar los problemas de comunicación entre el control y la gestión informática. Para conseguir este objetivo se ha optado por un sistema de programación libre de contacto basándose en Eventos. Estos eventos son acciones que se desencadenan por un hecho físico y son remarcables o importantes desde el punto de vista informático. Con este sistema se prescinde de la necesidad de establecer un mapa de comunicaciones entre la informática y el control y prácticamente toda la configuración se realiza desde la parte de control interactuando con la gestión informática allí donde es necesario y esta lo requiere para efectuar una correcta gestión.

Para lograr este trabajo se definen unas rutas de movimientos en la instalación, mediante las cuales se realicen los movimientos pertinentes cumpliendo a su vez los siguientes principios:

1. Los movimientos los desencadena el sistema de gestión.
2. El control realizará el movimiento siempre y cuando tenga la información necesaria para realizarlo, cuando no disponga de esta información debe ser la informática la que tome la decisión.
3. La informática debe ser notificada con todos los eventos que interesen a la misma, a efectos de realizar una gestión optima de la instalación.

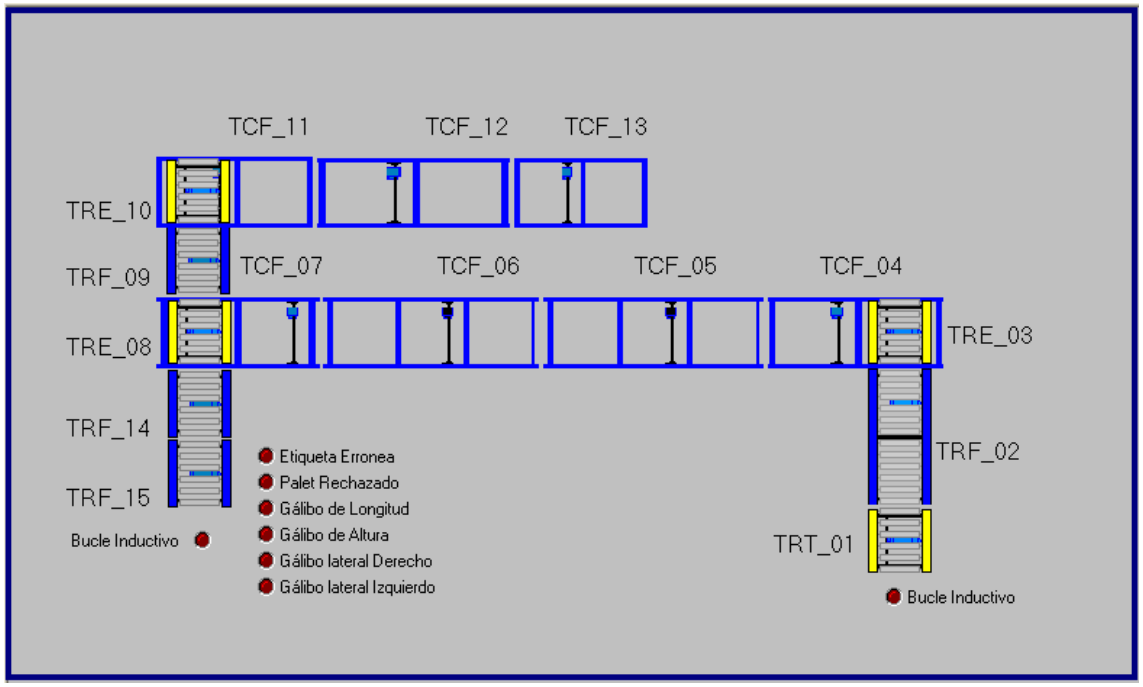
Con estos objetivos en mente, el responsable de la parte informática, de forma conjunta con la persona que vaya a desarrollar la aplicación de control, deben ponerse de acuerdo y definir los siguientes puntos:

1. *Trayectorias*: Recorridos posibles de las unidades de transporte por la instalación.
2. *Estaciones*: Puntos en los cuales los elementos de control deben de finalizar la orden que tengan en un momento dado y buscar otra para seguir avanzando.
3. *PIEs*: Puntos Identificativos de Entrada. Puntos hasta los que el palet llega mediante la aplicación de control sin que la gestión informática intervenga y en los cuales se realizarán los pertinentes controles e identificación y se notificará a la informática a fin de que se proceda a realizar la acción pertinente.

De forma similar se deben acordar los procedimientos que se invocan para efectuar cada una de las acciones.

Como la explicación teórica de estos apartados es imposible, se describirá el procedimiento con un ejemplo. Para ello se utilizará el layout de una instalación como base al ejemplo.

La instalación de ejemplo tiene el siguiente layout:



En este layout debemos indicar los siguientes transportadores como transportadores clave en la instalación:

- *TRT_01*: Este es el transportador en que los operarios del almacén depositarán los palets que entran en la instalación. En este punto no hay ningún elemento identificativo, con lo cual su mención aquí es anecdótica, sólo para mencionar que el avance desde este punto hasta el siguiente se realiza mediante el programa de control sin intervención de la gestión informática.
- *TRF_02:* Este transportador es el puesto de gálibo, en él hay un pórtico de gálibo para comprobar las dimensiones de los palets y un escáner para identificarlo, por lo tanto es el punto en que se notificará a la informática que tenemos un palet nuevo, indicándole su etiqueta, los gálibos que presenta (si presenta alguno) y cualquier dato adicional de interés. Obviamente, desde este punto no debemos mover el palet por motu propio, ese palet debe de tener un destino, que a priori (según el layout de la instalación) puede ser el puesto de rechazos (TRF_15) o bien el puesto de entrada al almacén (TCF_13). En todo caso, la disquisición de hacia donde debe dirigirse el palet tiene un componente físico (está claro que un palet con gálibo de longitud o un palet al que no se haya podido leer la etiqueta correctamente no se les puede mandar a la entrada del almacén) y un componente lógico (puede que el palet con etiqueta que se ha puesto en la entrada no esté dado de alta para la gestión informática, o bien puede que sea un palet de un producto que no deba de entrar en el almacén). Para que la decisión se pueda realizar siempre desde uno de los lados (control / gestión) este punto se definirá como *PIE* (Puesto Identificativo de Entradas), es decir notificará a la gestión que palet ha llegado y sus condiciones de gálibo junto con datos adicionales (si se desea). A su vez, parece claro que una vez notificada la llegada del palet a este punto, es la gestión informática

quien debe decidir su destino y comunicárnoslo, por lo que este transportador debe ser también un punto de *búsqueda de orden.*

- *TRF_15:* Este es el transportador de rechazos. A él irán dirigidos todos los palets que sean rechazados por motivos físicos o lógicos. Parece claro que, si el sistema de gestión ha enviado una orden para ir desde el transportador TRF_02 al TRF_15, se deba notificar a la gestión informática cuando el palet llega a este destino (bien para que, mediante radiofrecuencia, se envié a un carretillero a retirarlo, bien para que se avise en una pantalla que el palet sea retirado). Luego este es un *punto de finalización.*
- *TCF_13:* Este es el transportador de entrada al almacén. Una vez que un palet llegue a este punto, el transelevador puede recogerlo del transportador bien para ubicarlo en el almacén o bien para depositarlo en los transportadores de gravedad de salida. Parece claro que hasta este punto se llega desde la entrada por una orden del sistema de gestión y que en este punto se debe notificar al host la llegada del palet para que el transelevador venga a recogerlo cuando pueda. Es por tanto un *punto de finalización.*

Hasta aquí, todo resulta bastante lógico. Ahora vamos a describir como funcionaría el almacén. En el mismo tenemos un transelevador con un carro satélite. Los transelevadores en general recogen la mercancía en un punto y la depositan en otro punto, este es un principio lógico, pero se rompe en el caso en que al ir a depositar una unidad de transporte a una ubicación de almacén detecte que ya hay algo ubicado en este punto. En ese caso el problema es que se queda con la carga sobre el mismo (es lo que se denomina *Error de Depósito*) y la única orden que debería aceptar es una *Corrección de Depósito* (orden con origen transelevador y destino cualquier otro punto). Por lo tanto podríamos incluir en la lista de puntos clave de la instalación los siguientes:

5.2.1 Configuración de estaciones

- *ALMACÉN*: Este es el elemento en el cual se desea depositar la mercancía. Lógicamente es destino de las ordenes (ya que las unidades de transporte acaban en él) por lo tanto será un *Punto de Finalización*, pero además será *origen de transporte* ya que las salidas parten de él, así como las reubicaciones o trasvases.
- *SALIDAS*: En esta instalación en toda la altura 1 del almacén se hayan dispuestos unos caminos de rodillos de gravedad que identificamos globalmente como SALIDA. Este elemento es solamente destino de movimientos, ya que el transelevador no puede recoger la mercancía de los puestos de salida, por lo tanto solamente será un *Punto de Finalización*.
- *TRANSELEVADOR*: En el caso en que el transelevador se quede con una unidad de transporte en la cuna (por encontrarse el destino ocupado) parece claro que la orden siguiente que se genere debería tener el transelevador como origen de la misma, luego será *origen de transporte.* Sin embargo, hay que hacer notar que *TODAS LAS ORDENES SE FINALIZAN DESDE SU DESTINO TEÓRICO, AUNQUE NO CONSIGAN ALCANZARLO*, la diferenciación de si consigue alcanzarse el destino o no se realiza mediante el código de error al finalizar la orden.

En esta instalación no hay ningún punto más de movimiento que se pueda mencionar así que pasamos a comentar como se realizaría la configuración del

Agente de Transportes y como se debería realizar la comunicación con la informática.

Una vez definidos los puntos, se debe elaborar un documento indicando los nombres lógicos que se darán a cada uno de los elementos o puntos que hacen de *origen de transporte* o *punto de finalización*. Estos puntos se deben asociar con un procedimiento almacenado en PL/SQL que es el que el agente de transportes invocará de forma automática en el momento en el que uno de los eventos deseados ocurra.

Para esta instalación se definieron los siguientes los puntos lógicos o *Estaciones* de la siguiente manera:

Estaciones | Trayectorias | Procesar Ordenes | Averias | Codigo Generado

Nombre de Estación	Nro. Estación	Tipo	X	Y	Lado	Pasillo	Profundidad
PIE_01	1	3	0	0	0	0	0
RECH_01	1	4	0	0	0	0	0
ALM_01	1	0	*	*	*	*	*
SAL_01	1	10	*	*	*	*	*
TRANS_01	1	1	*	*	*	*	*
ENT_01	1	9	1	1	1	1	1

Los nombres utilizados han sido:

- *PIE_01:* Para el transportador TRF_02. Sus coordenadas se han definido como todo 0 ya que no es importante al no tener que ir ninguna máquina móvil a recoger la unidad de transporte a las mismas.
- *RECH_01:* Corresponde al transportador TRF_15. Sus coordenadas se han definido como todo 0 debido a las mismas razones que en el apartado anterior.
- *ALM_01:* Nos referimos con este nombre de estación al almacén. Será por tanto esta estación TODO el almacén con TODAS las ubicaciones de las que se dispone. Por dicho motivo parece claro que la posición no puede ser un elemento fijo (cada ubicación del almacén tendrá su posición), para definir en este caso que la posición no es fija y que por lo tanto *debe ser indicada al generar la orden* se recurre al carácter * (asterisco). Este indica que la coordenada debe venir indicada con la orden. Es de hacer notar que la representación de * es una convención para representar cualquier ubicación, y que no es el formato interno de almacenamiento por lo que sólo veremos los * cuando configuremos la pantalla. Una vez que los datos se almacenen, son sustituidos por –1, que es el formato de almacenamiento interno para el *. Por lo tanto no debe extrañarnos si al abrir el archivo por segunda vez vemos –1 donde hemos colocado *.
- *SAL_01:* Esta estación representa los 47 transportadores de gravedad que conforman la salida. Al igual que en el caso anterior, las coordenadas no son fijas sino que deben ser indicadas con la orden.
- *TRANS_01:* Se define la estación transelevador como tal ya que para resolver los errores de depósito debemos generar una orden para la

unidad de transporte que se encuentre en la cuna del mismo. Lógicamente sus coordenadas no son en ningún caso tenidas en cuenta ya que representarían coordenadas origen y parece ridículo que el transelevador se desplace a ningún origen si ya se encuentra con el palet en la cuna. En este caso han sido colocadas flotantes (es decir se aplicarán las que vayan en la orden), pero muy bien se podrían colocar a 0 ya que serán en todo caso despreciadas.

- *ENT_01:* Se define esta estación para el PIE (Transportador TRF_02). Tendrá las coordenadas a 0, ya que no tienen importancia al no tener que ir ningún elemento móvil hasta este punto.

Se debe hacer notar que cada estación definida debe tener un *número y tipo de estación*. El número de estación es solamente un indicador de orden en el caso en que existan más de una estación de un tipo. Por ejemplo si hay varios puestos de picking estos deberían numerarse comenzando en 1. El tipo indica que tipo de estación es a la que nos referimos. Históricamente se vienen utilizando los siguientes tipos de estación, aunque no en todas las instalaciones se han seguido estas convenciones. La convención recomendada es la siguiente:

NOMBRE	TIPO ID	DESCRIPCIÓN
ALM	0	Almacén propiamente dicho
TRASLO	1	Máquina transelevador
PK	2	Puesto de picking
PIE	3	Puesto de identificación de entradas
PS	4	Puesto de salida
REAC	5	Puesto de reacondicionamiento
RECH	7	Puesto de rechazos
MEP	8	Puesto de salida del picking con control de gálibo
ME	9	Mesa de entrada al almacén (carga transelevador)
MS	10	Mesa de salida del almacén (descarga transelevador)
MU	11	Mesa de ubicación
RMP	12	Máquina remontador
APL	13	Máquina apilador/desapilador
LANZ	14	Máquina lanzadera
PKI	15	Puesto de entrada del picking inverso
MP	16	Puesto de preparación de picking
MSC	17	Puesto de consolidación
PKE	18	Puesto de entrada del picking
ET	20	Puesto o estación de tránsito
CME	21	Puesto de entrada al pasillo desde un carrusel
MEF	22	Puesto de entrada a máquina enfardadora
MSF	23	Puesto de salida de máquina enfardadora
CROS	24	Puesto de salidas directas del PIE
APV	25	Máquina transportador de apilador de palets vacíos
CG	26	Puesto de control de gálibo
DOCK BUFFER	33	Pulmón de salidas
DOCK	34	Muelle
CH	35	Máquina carrusel horizontal
ALV	36	Máquina elevadora del almacén vertical
EQP	37	Equipamiento móvil (pistola, etc)

KIT ASSEMBLY	38	Estación para montaje de kits
TRUCK	40	Estación camión
RETURN	41	Estación para retorno de mercancía al almacén
RETENTION	42	Estación para acercamiento de pedido al muelle

5.2.2 Configuración de Trayectorias

A partir de esta primera configuración estamos en disposición de definir las diferentes trayectorias que se producen en esta instalación y cuál será la notificación a la informática.

Antes de entrar en la configuración de trayectorias que se ha utilizado en el ejemplo indicado se deben definir los siguientes conceptos:

- *Trayectoria Simple* → la que define el movimiento de una unidad de transporte desde una estación a otra, sin pasar por ninguna otra estación intermedia. En este caso la orden que genera la informática tiene origen la primera estación y destino la última estación. La orden que se ejecutará será exactamente la misma que la informática ha generado.
- *Trayectoria Compuesta* → la que define el movimiento de una unidad de transporte desde una estación hasta otra, pasando por ***n*** estaciones intermedias. La orden que genera la informática es desde la primera estación a la última. Implícitamente este tipo de trayectoria indica que la posición final ha sido decidida desde el momento en que se genera la orden. En estos casos el sistema va a descomponer la orden total generada por la informática en tantas órdenes parciales como estaciones intermedias existen, enviando cada orden parcial a su debido momento y generando automáticamente la siguiente cuando sea necesario hasta llegar a la estación final. Es posible recibir una notificación por cada punto intermedio por el que se pase.

Las trayectorias compuestas eliminan cierto trabajo que se realiza de forma "automática", pero generan una rigidez no apta en todas las situaciones. Por el contrario las trayectorias simples proporcionan mucha más flexibilidad de gestión, a costa de desplazar cierto trabajo a la informática.

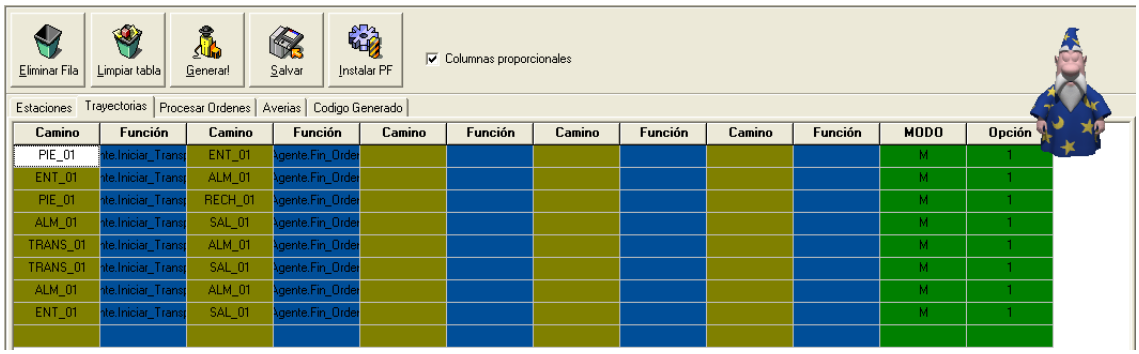
Las trayectorias, además, pueden tener un modificador de MODO de funcionamiento, que contempla 4 posibilidades:

- **M**: Trayectoria principal
- **B**: Trayectoria auxiliar en caso de bloqueo
- **A**: Trayectoria auxiliar en caso de avería
- **X**: Trayectoria auxiliar en caso de bloqueo o avería

Por lo general, no necesitaremos definir más que trayectorias principales.

El otro modificador aplicable a las trayectorias es llamado "Opción" y no es sino un peso que ayuda a elegir entre una u otra trayectoria en el caso de que haya varias opciones posibles en un instante.

Aclarados estos conceptos, abordemos ahora las trayectorias definidas para la instalación de ejemplo, son las siguientes:



Camino	Función	Camino	Función	Camino	Función	Camino	Función	Camino	Función	MODO	Opción
PIE_01	te.Iniciar_Trans	ENT_01	gente.Fin_Orde							M	1
ENT_01	te.Iniciar_Trans	ALM_01	gente.Fin_Orde							M	1
PIE_01	te.Iniciar_Trans	RECH_01	gente.Fin_Orde							M	1
ALM_01	te.Iniciar_Trans	SAL_01	gente.Fin_Orde							M	1
TRANS_01	te.Iniciar_Trans	ALM_01	gente.Fin_Orde							M	1
TRANS_01	te.Iniciar_Trans	SAL_01	gente.Fin_Orde							M	1
ALM_01	te.Iniciar_Trans	ALM_01	gente.Fin_Orde							M	1
ENT_01	te.Iniciar_Trans	SAL_01	gente.Fin_Orde							M	1

Como se puede apreciar, son todas las trayectorias lógicas posibles. Debe hacerse notar que acompañando a los diferentes puntos del camino aparece el nombre de una función. Este es el nombre de un procedimiento almacenado que puede estar dentro o no de un paquete y que en el caso de estarlo debe estar precedido por el nombre del paquete. En el caso en que se encuentre en un esquema diferente al del usuario de conexión de Galileo debe de ser especificado a su vez en la forma:

```
Esquema.Paquete.Procedimiento
```

El primer procedimiento que se define en un camino es el denominado procedimiento de inicio. Este se invoca *siempre que un transporte es elegido para ejecutarse (si la trayectoria es compuesta se invocará también en cada inicio parcial).* Cuando es invocado la orden ha pasado ya de pathfinder_msg_tr a pathfinder_transporte. Este procedimiento debe tener el siguiente prototipo:

```
PROCEDURE Iniciar_Transporte( n_Transporte      IN INTEGER,
                              n_Palet           IN INTEGER,
                              n_Error           OUT INTEGER,
                              v_Error           OUT VARCHAR2 );
```

Los siguientes procedimientos que se definen son los que se invocan en la finalización de las estaciones a las que acompañan, estos tendrán el siguiente prototipo:

```
PROCEDURE Fin_Orden(   n_Transporte    IN INTEGER,
                       n_Palet         IN INTEGER,
                       n_CodError      IN INTEGER,
                       v_Estacion      IN VARCHAR2,
                       n_Auxiliar      IN INTEGER,
                       n_Error         OUT INTEGER,
                       v_Error         OUT VARCHAR2);
```

Estos procedimientos deben devolver el valor n_Error a cero si desean sea un éxito la invocación o un valor distinto de cero si desean abortar la acción.

5.2.3 Configuración de Procesar Órdenes

Los puntos de búsqueda de ordenes y de notificación de llegadas de palet a la instalación PIEs son puntos con características especiales, ya que en ellos es

posible definir como se buscará una orden y como se notificará el evento de palet en PIE.

El esquema para definir uno de estos puntos siempre es el mismo:

5.2.3.1 Selección de la estación

Se introduce el nombre de red o dirección IP de la estación (computador) desde el que se quiere que se aplique un determinado procesado de órdenes. Este nombre es libre (es decir: se pueden poner nombres incorrectos), dado que puede ser interesante poder preparar la configuración para ordenadores que aun no estén disponibles, o bien preparar una sola configuración de agente de transportes para una instalación compleja con varias computadoras.

5.2.3.2 Seleccionar la máquina

Se selecciona la máquina desde la cual se va a buscar la orden (o notificar palet en PIE). Esta máquina no se puede teclear sino que la hay que seleccionar de un desplegable, es decir, no se permiten configuraciones incoherentes, solo se pude definir un punto de búsqueda o de palet en pie si la máquina está definida.

5.2.3.3 Identificación del procedimiento de búsqueda o PIE

En función de si lo que deseamos definir es un procedimiento de búsqueda o de palet en PIE los procedimientos a definir serán de la siguiente forma:

```
FUNCTION Buscar_Orden(  n_x          IN INTEGER,
                        n_y          IN INTEGER,
                        n_lado       IN INTEGER,
                        n_pasillo    IN INTEGER,
                        v_estado     IN VARCHAR2
                      ) RETURN INTEGER;
```

Como se puede apreciar la función devuelve un entero, que deberá ser 0 si no ha habido resultado en la búsqueda o el número de transporte deseado para su ejecución. Como parámetros de entrada la función tiene los siguientes:

1. *n_x:* Coordenada X en la que se encuentra la máquina (Normalmente de utilización real en el transelevador pero no en otros muchos casos). Representa la posición X de la máquina, aunque esto es una convención, si se acuerda entre control/informática otro significado `pues será este el significado. En caso de no utilizarse el parámetro entrará con el valor 0 (Valor que se decide desde control).
2. *n_y*: Coordenada Y en la que se encuentra la máquina (Normalmente de utilización real en el transelevador pero no en otros muchos casos). Representa la posición Y de la máquina, aunque esto es una convención, si se acuerda entre control/informática otro significado `pues será este el significado. En caso de no utilizarse el parámetro entrará con el valor 0 (Valor que se decide desde control).

3. *n_lado*: Lado en el que se encuentra la máquina, la utilidad de este parámetro es la que se quiera dar de mutuo acuerdo control/informática.
4. *n_pasillo*: Pasillo en el que se encuentra la máquina, este parámetro se suministra por control y pude significar esto o cualquier otra cosa que acuerde control/informática.
5. *v_estado*: Este parámetro refleja el estado de la máquina, sigue siendo de mutuo acuerdo control informática. El principio que se ha adoptado hasta el momento es que en el caso del transelevador se pasa 'S': Servicio, 'E': Error de depósito

Estos parámetros carecen de utilidad en muchos casos (transportadores), son de utilidad en otros muchos (transelevadores) y el significado de los mismo se acuerda entre control/informática.

Si lo que queremos definir es el procedimiento de PIE, este debe tener el siguiente prototipo:

```
FUNCTION Palet_En_PIE ( n_Flags      IN INTEGER,
                        v_Data       IN VARCHAR2
                      ) RETURN INTEGER;
```

Los parámetros son: los Flags de palet en PIE a acordar entre control e informática y los datos de la lectura (y adicionales si se desean) a acordar entre control e informática. Dado que control puede pasar 50 bytes para notificar palet en PIE, en la cadena v_Data pueden ir más datos que estrictamente los de la lectura.

5.2.4 Identificación del Tipo de Procedimiento

Mediante un desplegable se puede seleccionar si el procedimiento es de búsqueda de orden o de PIE.

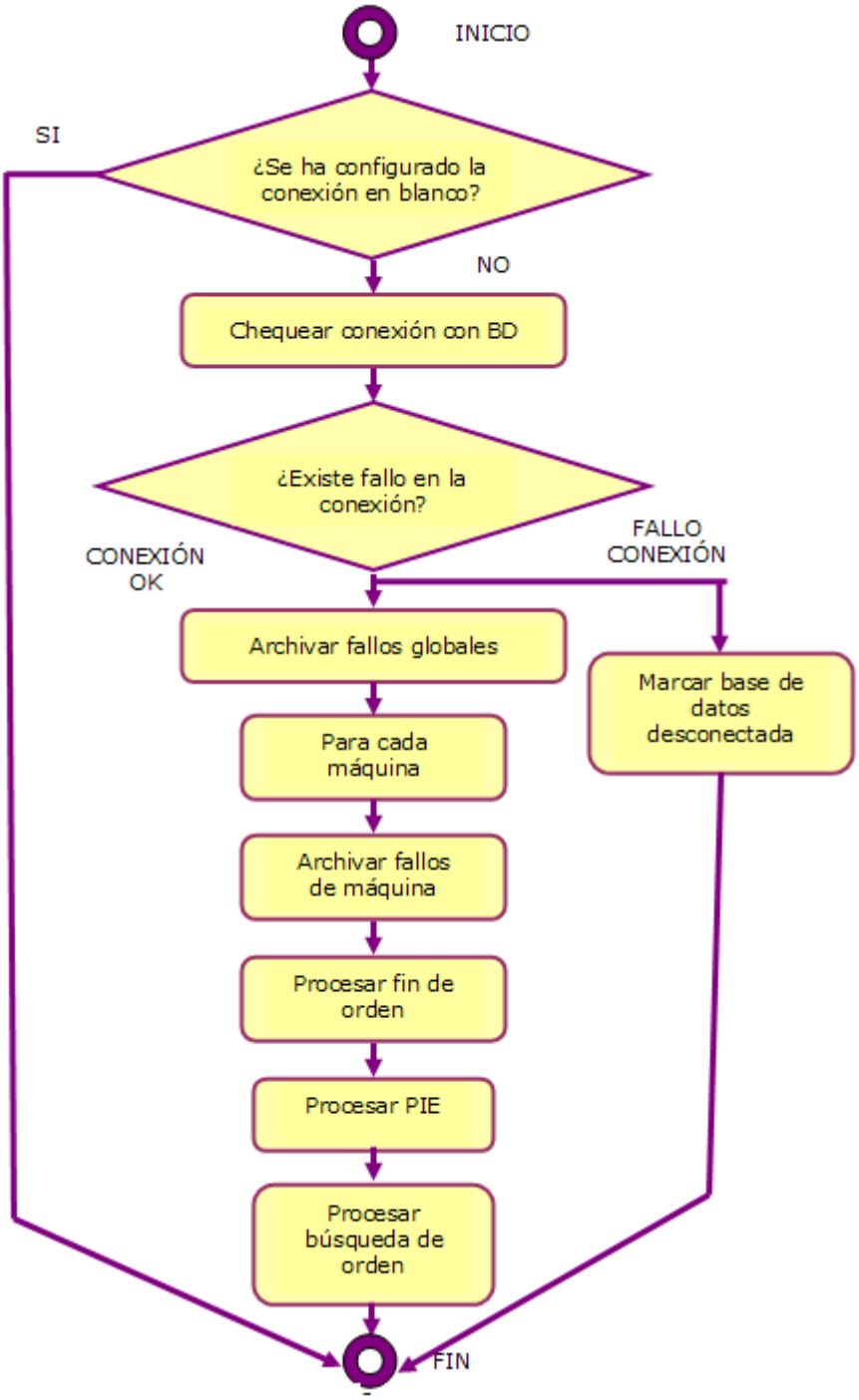
5.2.5 Generación de órdenes de transporte desde la parte informática

Para generar orden se utiliza la función `Generar_Orden`. Esta función esta sobrecargada y hay unos cuantos prototipos de la misma función. Estos prototipos existen por comodidad ya que a veces el destino no tiene coordenadas (al ser un destino con coordenadas fijas), o bien el origen o bien los dos, con lo cual hay prototipos que nos salvan de poner todos los parámetros. Como parámetro a comentar esta v_custom_data, que por defecto es NULL y que nos permite que la parte de control arrastre los datos que enviamos con el tracking. Puede arrastrar hasta 25 bytes.

5.3 Algoritmo de ejecución del Agente de Transportes

Para lograr una total comprensión de cómo funciona el agente de transportes se explicará en detalle el algoritmo de funcionamiento del mismo, de manera que se clarifiquen los conceptos. Es de vital importancia entender que el agente de transportes se interrelaciona con el thread de control (programa de control), con la base de datos y con las rutinas de gestión informáticas, por lo tanto el buen entendimiento de esto nos permitirá solucionar problemas de puesta en marcha.

Como principio general, el agente de transportes sigue la siguiente línea de ejecución:



Solo se puede llegar a comprender de forma completa el algoritmo explicando para las acciones fundamentales, es decir Palet en PIE, Fin de Orden y Búsqueda de

orden, la interacción entre las 3 capas del software, es decir Agente, Control y Programa de Gestión.

5.3.1 Procesado de la multi-operación

Hasta la aparición de Galileo IV, el programa de control solo podía ejecutar operaciones unitarias a través del agente de transportes. Es decir, solo podía ejecutar un fin de orden para un transporte en una máquina en un ciclo del agente de transporte. Lo mismo ocurre para los eventos y las búsquedas de orden, con la salvedad que estas últimas pueden solicitar más de un movimiento de cada vez. A partir de la aparición del agente de transporte 4.0, es posible que una máquina realice varias operaciones en un mismo ciclo. En un caso extremo se podrían realizar, mediante una sola operación hasta 10 eventos, 10 fines de orden y una búsqueda de 10 ordenes. De esta manera, el número de operaciones a realizar se minimiza, evitando así retrasos producidos por la sobrecarga del agente de transporte.

5.3.1.1 Slots

La multi-operación se apoya en el conceto de slot. Un slot no es mas una zona de memoria en la que el programa de control puede solicitar un fin de orden o un evento. El numero de slot disponibles es igual al número de trackings permitidos por máquina, en este caso 10. De esta manera, aparte de las zonas clásicas de búsqueda de orden, finalización y evento, ahora se dispone también de 10 slots que pueden ser utilizados para la multioperación.

El uso de los slots se tiene que realizar de manera secuncial. Es decir, el programa de control tiene que utilizar el slot 1, luego el 2 y así sucesivamente. No se puede utilizar, por ejemplo, el slot 5, sin utilizar los anteriores, ya que el agente de transportes los utiliza también de forma secuencial.

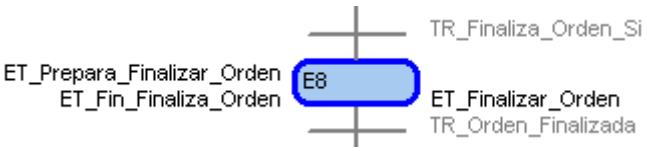
Las operaciones a realizar para la multioperación son las mismas que para las operaciones clásicas, salvo que las funciones van precedidas por el prefijo MultiOP y aceptan un número de slot. Por ejemplo, la operación SetMultiOPIISTransport, establece el número de transporte en el evento, de forma análoga a como lo hace la operación SetIISTransport clásica.

Es importante tener en cuenta que aunque existen 10 slots, estos solo se aplican a los eventos y a los fines de orden. La zona de búsqueda de orden de la multioperación es única, ya que se permita desde ella la obtención de hasta 10 movimientos.

5.3.2 Procesado de Fines de Orden

Como todas las operaciones gestionadas por el agente de transportes estas se inician desde Control y son seguidas por la informática. El algoritmo detallado es el siguiente:

Siempre que la parte de control quiere finalizar una orden utiliza una estructura idéntica existiendo una etapa de la siguiente forma:



Normalmente el código de finalización de orden que se debe aplicar es el siguiente:

1. Acción de entrada

Función *ET_Prepara_Finalizar_Orden*

```
//================================================================
// Prepara la finalización de orden. Por defecto se finalizará el
//tracking 1
//================================================================

  this->PrepareCommandToFinish( 1, 0, 0 );

// Código para poder sobreescribir los datos para el fin de orden si
// es necesario

  this.inline.p_Code_PreparaFinOrden;

//================================================================
```

Como se aprecia en la función se indica que los Datos del Tracking 1 serán los utilizado para finalizar la orden. De este Tracking se extrae la siguiente información:

1. Número de Transporte
2. Número de WTU
3. Número de Entidad que finaliza
4. Tipo de Entidad que finaliza

Y adicionalmente el segundo y tercer parámetros de la función son el código de error con el cual deseamos finalizar y los datos auxiliares con los que deseamos finalizar.

Cabe destacar que normalmente (al igual que ocurre con el Monitor de Transportes) si el Tracking acaba por llegar a un punto incorrecto (a otra estación) y esta finaliza, lo hará con los datos del tracking (que son incorrectos). Es deber del programador de control el controlar esta situación indicando a través del código de error o del auxiliar este hecho a la informática para que tome las acciones oportunas.

2. Acción de Etapa

Función *ET_Finalizar_Orden*:

```
//================================================================
```

```
// Indica al Agente de Transportes que los datos preparados
//anteriormente(en la acción anterior de la etapa) deben utilizarse
//para finalizar la orden
//=============================================================

  this->TerminateCommand();

//=============================================================
```

Este código activa un flag que indica al agente de transportes que la máquina en cuestión está solicitando finalizar la orden. Este flag es el detectado por el agente de transportes para empezar a procesar un fin de orden.
En este momento el control se realiza por el agente de transportes que comenzará la siguiente ejecución:

1. Se verifica que la estación que finaliza exista en la definición de estaciones del Agente de Transportes. Si la dirección no existe se finaliza la orden a la parte de control. No se notifica a la informática el fin de orden ante la imposibilidad de identificar el procedimiento de finalización.
2. Si la estación existe se verifica que el transporte exista. Si el transporte no existe se localiza la primera trayectoria parcial que finalice en esta estación. Utilizando el procedimiento de finalización de esta estación se señala el fin de orden a la informática para que actúe en consecuencia (normalmente si no existe transporte este palet está perdido y se debería generar una orden para extraerlo de la instalación).
3. Si el fin de orden procede de una estación que no corresponde con la orden en curso se finaliza la orden a control pero no se hace nada más, con lo cual el Transporte finalizará en este punto y no continuará su desplazamiento.
4. Se verifica que exista una trayectoria parcial para llegar a destino.
5. Si la estación que finaliza no es la última de la trayectoria se comprueba que exista una trayectoria parcial que vaya a ese destino desde el punto de Fin de Orden.
6. Se ejecuta el procedimiento almacenado de usuario. El usuario puede querer generar ordenes desde este procedimiento invocando a PATHFINDER.Generar_Orden().
7. Si el procedimiento de usuario devuelve un valor distinto de cero se cancela la ejecución, es decir no se finaliza la orden a la parte de control.
8. Si el procedimiento no falla se inserta el fin de orden en pathfinder_msg_hs.
9. Si la estación que finaliza es la última de la trayectoria o bien el código de error es distinto de cero
    a. Se inserta en el registro en pathfinder_hist_transporte.
    b. Se elimina el registro de pathfinder_transporte.

10. En caso contrario (no es la última estación de la trayectoria y el código de error es cero).
    a. Se extraen los datos de la siguiente estación verificando que las coordenadas fijas / flotantes son correctas y las estaciones existen.
    b. Se inserta el registro en pathfinder_hist_transporte.
    c. Se inserta en pathfinder_msg_tr el nuevo registro de transporte.
    d. Se elimina el registro de pathfinder_transporte.

El Agente de Transportes solamente realiza *commit* si el retorno del procedimiento es 0 o bien –--20012 (No existe transporte). En los demás casos no se hace *commit* y no se finaliza a control el fin de orden.
Si el Agente de Transportes realiza *commit* lo siguiente que hacer es subir el flag a control para que el programa de control se entere del procesamiento del fin de orden. Esto se puede ver en la pantalla de edición de tracking etiquetado como "Flag Fin de Orden Recibido" (véase imagen anterior). Desde el punto de vista de control este es el momento donde se coordina la transición mostrada antes y que tendrá el siguiente código:

3. Transición

Función *TR_Orden_Finalizada:*

```
//================================================================
// Comprueba si la orden se ha finalizado por el AT
//================================================================

  return this->IsCommandTerminated();

//================================================================
```

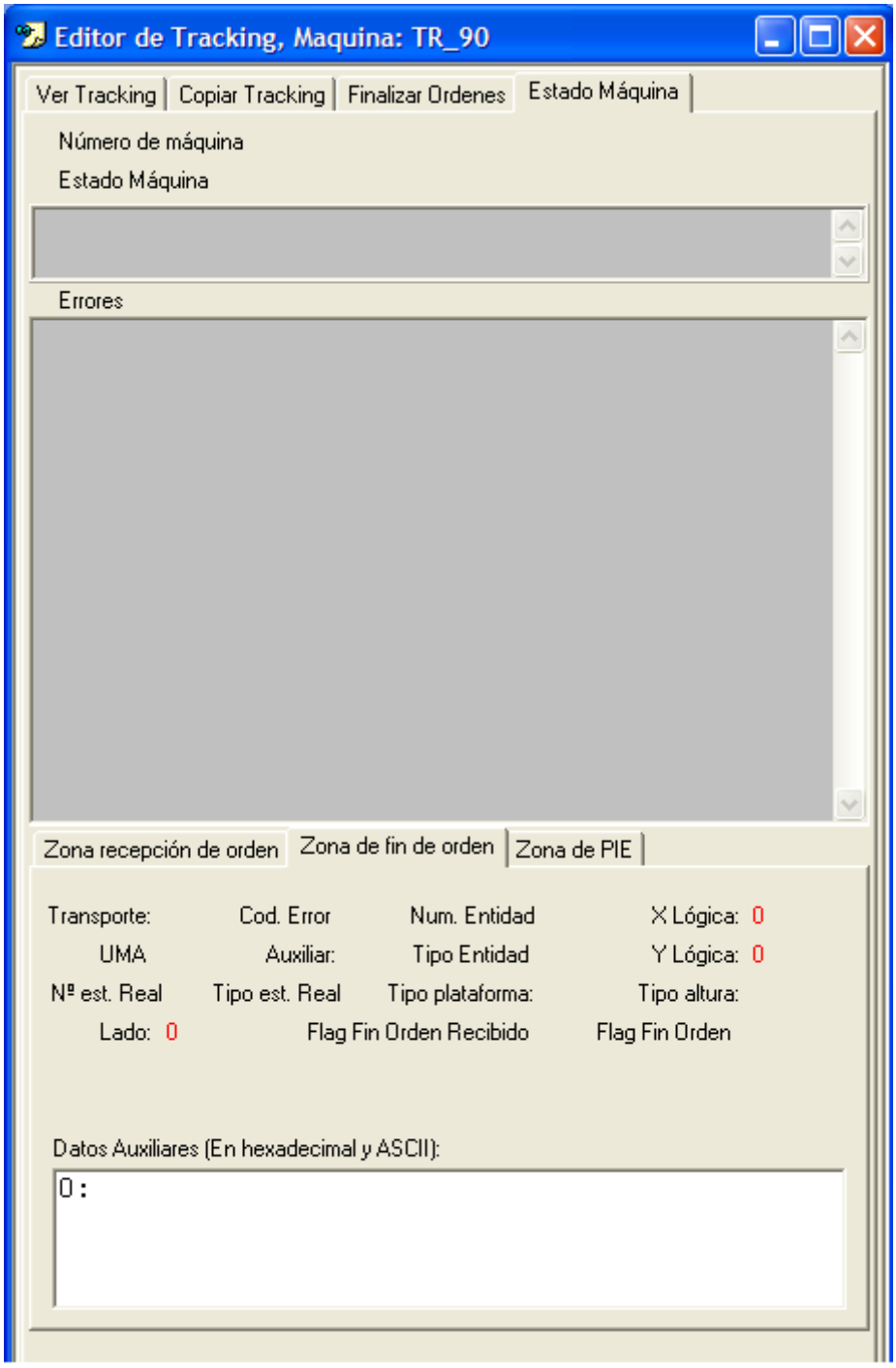
4. Acción de Salida

La etapa de salida para bajar el flag, tendrá el siguiente código:

Función *ET_Fin_Finalizar_Orden:*

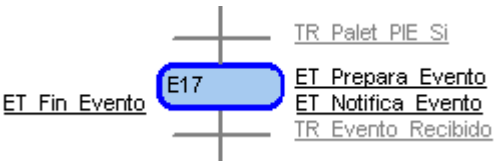
```
//================================================================
// Indica al AT que debe abandonar la finalización de orden
//================================================================

  this->ExitTerminateCommand();

//================================================================
```

Durante la operativa de fin de orden pueden plantearse errores. Estos errores se pueden visualizar fácilmente desde el editor de tracking. Esta pantalla mostrará toda la información relativa a lo que está haciendo el Agente de Transportes, los errores que genera la informática y el estado de los flags.



5.3.3 Proceso de Procesamiento de PIE

El procesamiento de PIE es similar en estructura al Fin de Orden. El proceso seguido es el siguiente:
Desde la parte de control la estructura de procesamiento de PIE tiene siempre una forma similar a esta:



A continuación describiremos lo que ocurre durante esta ejecución.

1. Acción Anterior

Función *ET_Prepara_Evento:*

```
// ====================================================================
// Se preparan los datos necesarios para realizar un evento de PIE
// ====================================================================

// Lectura de escáner que se realiza sobre la función de datos
// de PIE teniendo en cuenta el valor en la función de status que
// puede tener los siguientes valores:
// p_MW_StatusEscaner:
//                    = 1 -> Código de barras correcto
//                    = 2 -> Error lectura
//                    = 3 -> Basura en puerto de comunicaciones
//                    = 4 -> Desbordamiento
// Almacena el valor de Status de Scanner(si no existe escaner siempre
// vamos a tener 0)

  if ( this.p_C_Escaner->GetStatus () != 0 )
    this.p_MDW_StatusEscaner = this.p_C_Escaner->GetStatus ();
  dword i;
  i = 0;
  for ( i = 0; i < 40; i ++ )           // 40   --> 80
  {
    this->SetEventData ( i+10, 0 );     // i+10 --> i
    this->SetTrackingData ( i+10, 0 );
  }
  if ( this.p_MDW_StatusEscaner == 1 )
// El tope máximo de caracteres que podríamos llegar a leer suponemos
// que es 50. El fin de la lectura vendrá definido por que la
// adquisición del último caracter sea distinta de "0".
// IMPORTANTE!!!
// Si se amplia la lectura de la etiqueta porque sea requerido en una
// instalación hay que tener cuidado con su tamaño en la escritura de
// los datos en el custom data, ya que su limitación es mucho menor
// que en la zona PIE y puede generar errores en Galileo.
  {
    for ( i = 0; i < 40; i ++ )                 // 40   --> 80
    {
      if ( this.p_C_Escaner->GetData(i) != 0 )
      {
       this->SetEventData(
                          ( i+10 ),
                           this.p_C_Escaner -> GetData ( i )
                         );
       this->SetTrackingData(
```

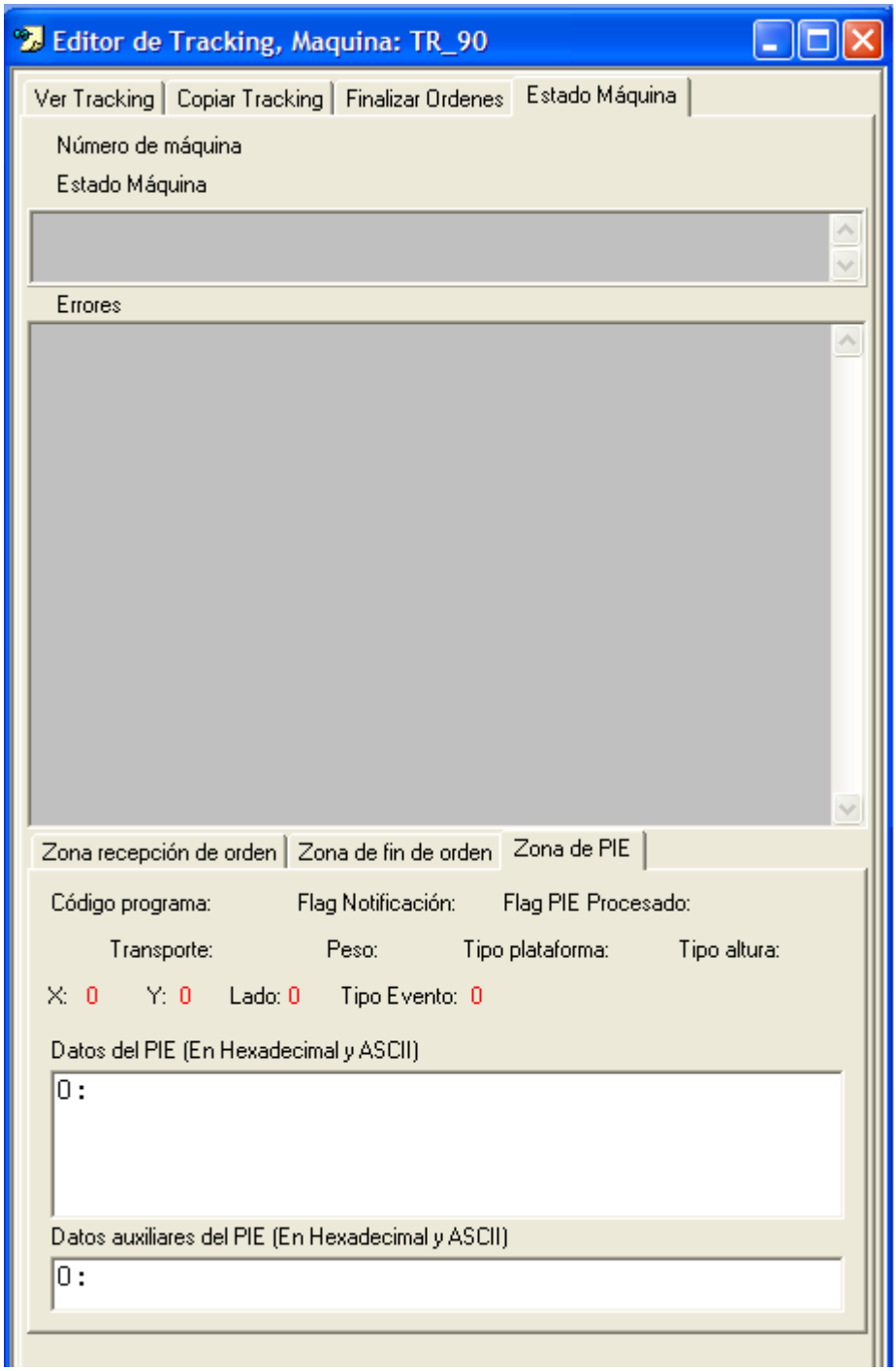
```
                                ( i+10 ),
                                 this.p_C_Escaner -> GetData (i )
                               );
        }
        else
          break;
      }
    }
```

```
// Se codifica una DWORD con los defectos de gálibo del palet para
// poder dárselos al Agente de Transportes
// ================================================================
// MDW_PaletPIE
// - Bit 0 -> Defecto en los huecos del palet.
// - Bit 1 -> Defecto de larqueros del palet.
// - Bit 2 -> Gálibo de exceso de altura.
// - Bit 3 -> Gálibo lateral derecho.
// - Bit 4 -> Galibo lateral izquierdo.
// - Bit 5 -> Gálibo por desplome del palet adelante.
// - Bit 6 -> Gálibo por desplome del palet atras.
// - Bit 7 -> Exceso de peso en el palet.
// - Bit 8 -> Lectura erronea en la etiqueta del palet.

// -------------- Bit 0 ----------------------
if ( this.p_M_ControlHuecos )
 this.p_MDW_PaletPIE = this.p_MDW_PaletPIE | 0x00000001;
// -------------- Bit 1 ----------------------
if ( this.p_M_ControlTacos )
 this.p_MDW_PaletPIE = this.p_MDW_PaletPIE | 0x00000002;
// -------------- Bit 2 ----------------------
if ( this.p_M_GaliboAltura )
 this.p_MDW_PaletPIE = this.p_MDW_PaletPIE | 0x00000004;
// -------------- Bit 3 ----------------------
if ( this.p_M_LateralDerecho )
 this.p_MDW_PaletPIE = this.p_MDW_PaletPIE | 0x00000008;
// -------------- Bit 4 ----------------------
if ( this.p_M_LateralIzquierdo )
 this.p_MDW_PaletPIE = this.p_MDW_PaletPIE | 0x00000010;
// -------------- Bit 5 ----------------------
if ( this.p_M_GaliboAdelante )
 this.p_MDW_PaletPIE = this.p_MDW_PaletPIE | 0x00000020;
// -------------- Bit 6 ----------------------
if ( this.p_M_GaliboAtras )
 this.p_MDW_PaletPIE = this.p_MDW_PaletPIE | 0x00000040;
// -------------- Bit 7 ----------------------
if ( this.p_M_ExcesoPeso )
 this.p_MDW_PaletPIE = this.p_MDW_PaletPIE | 0x00000080;
// -------------- Bit 8 ----------------------
if ( this.p_MDW_StatusEscaner != 1 && this.p_M_ExisteEscaner)
 this.p_MDW_PaletPIE = this.p_MDW_PaletPIE | 0x00000100;

// Procesa el palet en PIE con los datos recogidos anteriormente
// Para evitar el utilizar el valor "0" como palet OK, es decir aquel
// que no tiene ningún valor, mandamos en caso de palet OK el valor en
// decimal 65536
// ================================================================
```

```
  if ( this.p_MDW_PaletPIE == 0 )
    this.p_MDW_PaletPIE = 65536;  // Palet sin defectos de gálibo

// Preparación del evento Galileo que se enviará al SGA
// ===================================================

  this->SetEventStationNumber(this.p_MB_NumeroEstacion);
  this->SetEventStationType(this.p_MB_TipoEstacion);
  this->SetEventTransport(this.inline.p_Code_NumeroTransporte);

// Tipo de evento:
// 0 == PIE
// !0 == el evento realiza antes de procesarse un fin de orden sobre
// el transporte indicado.
// 1-> Palet en pie.
// 2-> Pulsador


this->SetEventType(this.inline.p_Code_TipoEvento);

// Datos de gálibo

this->SetEventFlags(this.p_MDW_PaletPIE);
// Peso
this->SetEventWeight(this.p_MDW_PesoActual);
// Tipo de altura
this->SetEventHeightType(this.p_MB_TipoAltura );
// Tipo de contenedor
this->SetEventPlatformType(this.inline.p_Code_TipoContenedor);
//Constantes de posicion
this->SetEventX(0);
this->SetEventY(0);
this->SetEventSide(0);

// Se insertan en "custom data" los valores enviados al SGA para que
// puedan ser visualizados a través del editor de Tracking

this->SetTrackingData ( 1, this.p_MB_TipoAltura );
this->SetTrackingData ( 1, this.p_MDW_PesoActual );

// Código para poder añadir o sobreescribir los datos que son
// requeridos por la informática en el evento

  this.inline.p_Code_PreparaDatosEvento;

// ================================================================
```

El procedimiento prepara los datos del área de PIE con la lectura del escáner. Se pueden componer estos datos con más valores, por ejemplo Lectura-Peso-Altura. Debemos tener en cuenta que estos datos deben conformar una cadena (cuyo terminador es cero) y que la informática debe estar enterada de cuál es el formato de los datos para procesarlos adecuadamente.

Todos estos flags y datos pueden observarse desde el editor de tracking, visualizando la máquina correspondiente, seleccionando la solapa "Estado Maquina" y después la solapa "Zona de PIE", tal y como se muestra en la siguiente imagen:



2. Acción de Etapa

Función *ET_Notifica_Evento:*

Durante la acción de etapa el código que se ejecutará es algo similar al siguiente:

```
// ====================================================================
// Notificación del evento galileo al SGA
// ====================================================================

  this->NotifyEvent(1);

// ====================================================================
```

Con este código estamos diciendo a la informática que Flag de Palet en PIE estamos notificando (téngase en cuenta que este flag es un valor entero y que su significado es un convenio entre control e informática).

Indicamos también el número y tipo de estación que da el PIE (no es realmente imprescindible). Y se invoca el procedimiento NotifyIIS() que es el que realmente sube el flag de notificación para que el Agente de Transportes se pueda enterar del deseo de notificar un PIE.

En este momento que el Agente se ha enterado de la Notificación lo que hace es lo siguiente:

1. Comprueba que la estación en cuestión tenga definido un PIE
2. Si los datos con que se invoca el procedimiento almacenado (datos de la zona de PIE) son NULL se cambia por el valor '0', si no, se utiliza directamente el contenido de la zona de PIE.
3. Se ejecuta el procedimiento almacenado de usuario configurado. A no ser que queramos que la unidad de transporte sea retirada manualmente del PIE, este procedimiento deberá llamar en algún momento a alguna de las funciones Genera_Orden del paquete PATHFINDER, a fin de generar una orden que de lugar al inicio del movimiento.
4. Si el procedimiento retorna valor 0 se realiza un *commit* y se notifica el procesado a control, si no, se hace rollback y no se continúa el protocolo con control. Esto quiere decir que el procedimiento de usuario debería siempre retornar 0 y si algo no le gusta generar la orden correspondiente a rechazos o la estación pertinente.

La bajada de flag por parte del agente de transportes se comprueba mediante la transición por parte del programa de control:

3. Transición

Función *TR_Evento_Recibido:*

```
// ====================================================================
// El evento ha sido recibido por el SGA
// ====================================================================

  return this->IsEventFlagReceived();

// ====================================================================
```

Mediante este procedimiento el programa de control comprueba que el Agente de Transportes haya bajado el flag de palet en PIE.

Si esto ocurre se invocará el procedimiento de salida de la etapa.

4. Acción de Salida

Función ET_Fin_Evento

```
// ================================================================
// Finaliza el protocolo de notificación del evento
// ================================================================

  return  this->ExitNotifyEvent();

// ================================================================
```

A partir de este momento el proceso se ha finalizado y el Agente de Transportes bajará su flag de coordinación.

5.3.3.1 PIEs (eventos) con el agente de transporte 3.1

Con la aparición del agente de transporte 3.1, la gestión de los PIEs se ve modificada. En concreto, desparece el concepto de PIE como tal, para ser sustituido por el de "evento". Un PIE es por tanto, un determinado evento, como lo puede ser también la pulsación de botón o cualquier otra situación que queramos informar a la informática.

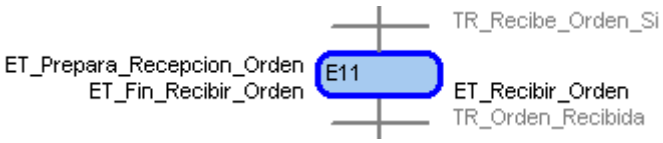
Las funciones utilizadas para la gestión de eventos en 3.1 son análogas a las utilizadas para la gestión de PIEs en 3.0. Al igual que existe un método *ExitNotifiIIS*, en la versión 3.1 existe un método *ExistNotifyEvent*. La utilización de estos métodos, es similar a los anteriores, cambiando únicamente el método *NotifyIIS*. Al ser el PIE un clase determinada de evento, es necesario indicar el código de evento cuando se quiera notificar. Por ejemplo, la notificación de un PIE se podría realizar de la siguiente manera:

```
this->SetEventFlags ( 1 );
this->SetEventStationNumber ( 1 );
this->SetEventStationType ( 3 );
this->NotifyEvent(1);
```

El código de evento asociado a un PIE es 1.

5.3.4 Proceso de Búsqueda de Orden

El proceso de búsqueda de orden es aquel por el cual las máquinas físicas buscan una orden para realizar un movimiento en determinados momentos de funcionamiento.
El proceso se inicia desde control y se sigue por el agente de transportes y la informática de usuario. El programa de control normal para realizar la búsqueda de orden será el siguiente:



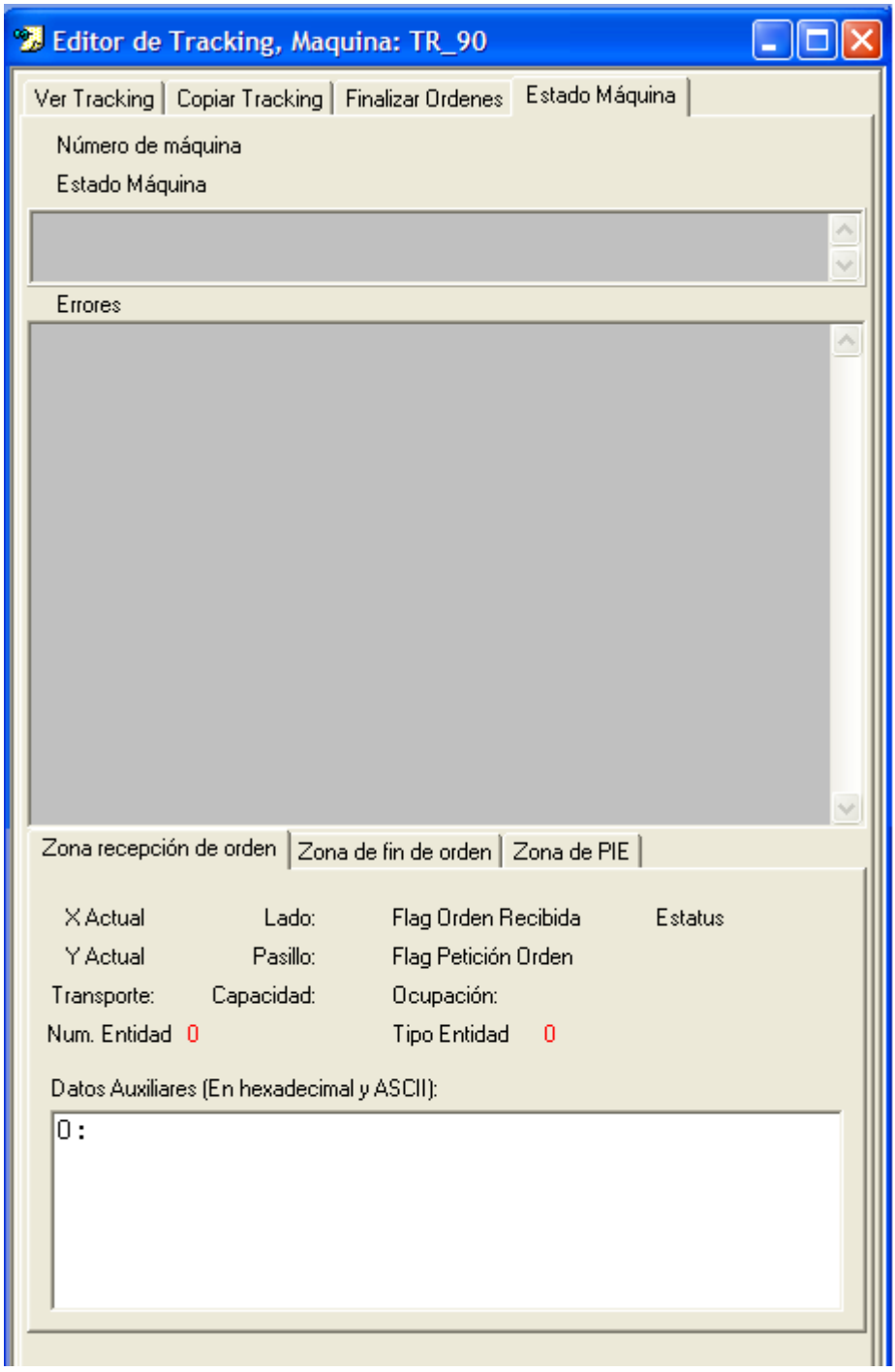
En este fragmento de programa el código de usuario sería el siguiente:

1. Acción de Etapa

Función *ET_Recibir_Orden:*

```
// ============================================================
// Activa la señal al Agente de Transportes para que se efectue la
// recepción de orden con los datos indicados
//============================================================

  this->SetCurrentStationNumber (this.p_MB_NumeroEstacion);
  this->SetCurrentStationType (this.p_MB_TipoEstacion);
  this->SetCurrentTransport (0);
  this->SetCurrentX(this.p_MDW_UbicacionX);
  this->SetCurrentY(this.p_MDW_UbicacionY);
  this->SetCurrentSide(this.p_MDW_UbicacionZ);
  this->SetCurrentAisle(this.p_MB_Pasillo);
  this->SetCurrentState(this.p_MW_ActEstadoTraslo);
  this->SetCurrentCapacity(this.inline.p_Code_Capacidad);
  this->SetCurrentOccupation(this.inline.p_Code_Ocupacion);

  this->ReceiveCommand();

// ============================================================
```

Todo esto se puede observar desde el editor de tracking, concretamente los flags y datos comentados son visibles en la solapa “Zona Recepción de orden” incluida a su vez en la solapa “Estado Máquina”:



Los métodos iniciales `SetCurrent…` son métodos para preparar información que debe ser luego procesada por la informática. Todos los valores representan estados que la informática debe tener en cuenta para decidir que orden es la óptima para que la máquina la ejecute. Inicialmente se ha considerado que los valores a tener en cuenta son las posiciones X, Y, Lado, Pasillo y Estado, pero esto no es más que una sugerencia de funcionamiento. Por ejemplo si el que busca orden es un transportador, no tiene mucho sentido el indicar la posición, sin embargo si puede

tener sentido que la parte de control indique cuantos palet se encuentran en determinado transportador, esto se puede hacer indicando este número en la coordenada X por ejemplo. Es decir estos estados deben ser tomados como una forma de indicar información a la gestión informática y que su significado debe ser un convenio entre informática y control.

NOTA: *Debe tenerse en cuenta que el método preferido para comunicar información que solo es utilizada para la búsqueda de orden es este, no la invocación de procedimientos almacenados para actualizar estados.*

El método `ReceiveCommand` es el que realmente sube el flag para informar al Agente de Transportes de que se desea recibir una orden. Una vez subido el Flag el Agente trata de buscar la misma a través del procedimiento almacenado que el usuario configura a tal efecto. La ejecución del Agente de Transportes realiza las siguientes tareas:

1. Comprueba que exista un procedimiento almacenado definido para realizar la búsqueda de orden.
2. Invoca el procedimiento almacenado con los estados que la parte de control notifica.
3. Si el número de transporte que la informática devuelve es 0 pues no se hace nada (no hay transporte).
4. Si existe un transporte se inserta la orden en pathfinder_msg_tr.
5. Se invoca el procedimiento de inicio de transporte definido por usuario.
6. Si este procedimiento no devuelve error se notifica la orden a control y se hace commit, si no se hace rollback.

NOTA: *Hay que tener en cuenta que la búsqueda de orden que se realiza en el procedimiento almacenado de usuario debe tener en cuenta el siguiente caso: Si hay varias máquinas que por sus procedimientos puedan seleccionar la misma orden, entonces el cursor que el usuario abra para realizar la búsqueda debe abrirse FOR UPDATE, de manera que el registro quede bloqueado y no pueda darse el caso de que dos máquinas intenten buscar la misma orden. También debe tenerse en cuenta el hecho de que el Agente de Transportes no utiliza el sistema Guardián del Monitor de Transportes. Es el usuario el que debe realizar la acción oportuna si la búsqueda de orden resulta con que hay varias ordenes posibles y solo debería haber una (ordenes antiguas perdidas en la instalación). Como mínimo debería darse error y notificar porque para que desde el editor de Tracking se pueda ver el problema.*

Llegados a este punto la transición de búsqueda de orden que se ejecuta desde control debería retornar true. Su código es el siguiente:

2. Transición

Función *TR_Orden_Recibida:*

```
// ================================================================
// Devuelve true en el caso en que el Agente de Transportes indique
// que se ha recibido orden.
```

```
// ===========================================================================

  return this->IsCommandReceived();

// ===========================================================================
```

En caso de que la transición sea cierta se ejecutará la acción de salida

3. Acción de salida

Función *ET_Fin_Recibir_Orden:*

```
// ===========================================================================
// Señala al Agente de Transportes que abandone la recepción de orden
// ===========================================================================

   this->ExitReceiveCommand();

// ===========================================================================
```

Durante la instalación del Designer se instalan en el directorio Samples varios ejemplos. En el directorio Referencias\1 se encuentra un ejemplo básico con todo lo tratado en este capítulo, incluyendo el programa de control, la configuración del agente de transportes y el paquete PL/SQL de ejemplo.

5.3.4.1 Búsquedas de orden con el agente de transporte 3.1

El agente de transportes 3.1 permite la recepción de varias órdenes simultáneamente. Esto es posible ya que en la búsqueda es posible ahora indicar la capacidad de la estación y su ocupación actual. Con estos datos, la informática devolverá tantas órdenes como le sea posible. Estas órdenes serán almacenadas en los distintos trackings de la máquina. Por ejemplo, para indicar que una estación es capaz de recibir hasta 4 trackings y que actualmente ya tiene dos ocupados, se puede escribir lo siguiente:

```
this->SetCurrentCapacity(4);
this->SetCurrentOccupation(2);
```

5.3.5 Cancelación de órdenes desde informática

Es posible que en algún momento, la informática necesite cancelar una orden. La cancelación de la orden por parte de la informática necesita ser confirmada por el sistema de control.
La manera de realizar esto desde la parte de control es comprobando en la etapa anterior del secuenciador y cíclicamente si alguno de los trackings actuales de la máquina tiene un evento de cancelación.

El código estandarizado para ello dentro de todos los secuenciadores del repositorio es el siguiente (ET_AcciónAnterior):

```
//================================================
// Gestión de cancelación de órdenes desde el SGA
//================================================
```

```
  this->SelectTracking(1);

  this.p_MDW_NumeroTransporte = this->GetTransportNumber();

  if (
      this.p_M_Manual
      &&
      g_System->TransportHasCancelEvent(this.p_MDW_NumeroTransporte)
      &&
      this.p_MDW_NumeroTransporte != 0
     )
  {
   g_System -> SetCancelEventResponse(this.p_MDW_NumeroTransporte,1);
   this->ClearTracking(1);
   this->Restart();
   this->EvalFailure( 60, false , this.p_M_Defecto );

   if (
       this.ant1.p_MDW_Posterior == this->Number()
       &&
       this.ant1->TrackingCount() == 0
      )
   {
    this.ant1->Restart();
   }
   if (
       this.pos1.p_MDW_Anterior == this->Number()
       &&
       this.pos1->TrackingCount() == 0
      )
   {
    this.pos1->Restart();
   }
  }

 // Si no es posible cancelar la orden se informa el SGA

 else if(
       g_System->TransportHasCancelEvent(this.p_MDW_NumeroTransporte)
        )
  {
   g_System -> SetCancelEventResponse(this.p_MDW_NumeroTransporte,0);
  }

// ===================================================================
```

En este ejemplo en cuestión, cualquier orden de cancelación que nos manden será cancelada. Ante una orden de cancelación, el programa de control puede tomar dos decisiones:

1. Cancelar la orden. Deberá invocar el método *SetCancelEventResponse* con estado 1.

2. No cancelar la orden. Si no quiere, o no puede, cancelar la orden en ese momento, el programa de control puede invocar el método *setCancelEventResponse* con estado 0, o no invocar este método. En ambos casos, el comportamiento es el mismo. La no cancelación de la orden por parte de informática no detiene la cancelación en la informática, sino que ésta seguirá enviando la orden de cancelación hasta que la respuesta sea positiva.

5.3.6 Códigos de error devueltos desde el paquete PathFinder

Los procedimientos almacenados del paquete PATHFINDER reservan ciertos números de error que son usados para indicar fallos en algún momento. Como regla general, el programador del sistema de gestión (informática), debería respetar estos números, y en caso de necesitar devolver códigos de error desde sus procedimientos, hacerlo a partir del 20030. A continuación se presenta una tabla con todos los posibles errores que pueden devolver desde los procedimientos del paquete PATHFINDER, indicando el o los procedimientos que pueden generarlo y una breve explicación de la causa que los origina.

Código de Error	*Generado en Procedimiento*	*Explicación*
-20001	*Insertar_Procesado_Ordenes* *Insertar_Procesado_PIES*	Ya existe una entrada para la combinación máquina/tarjeta/secuenciador que se pretende insertar.
-20002	*Insertar_Procesado_Ordenes* *Insertar_Procesado_PIES*	El procedimiento a insertar es NULL
-20003	*Insertar_Det_Trayectoria*	No existe una trayectoria maestra para una inserción en det_de_trayectoria
-20004	*Genera_Orden*	El Origen Introducido es desconocido, no esta definido en la tabla estaciones
-20005	*Genera_Orden*	El Destino Introducido es desconocido, no esta definido en la tabla estaciones
-20006	*Buscar_Orden* *Genera_Orden*	No se ha definido ninguna trayectoria para Ir de Origen a Destino
-20007	*Genera_Orden*	Aunque la Trayectoria existe es imposible reconstruirla a partir de detalles trayectoria
-20008	*Genera_Orden*	La entidad Origen al llamar a esta función debe tener las coordenadas fijas
-20009	*Genera_Orden*	La entidad Destino al llamar a esta función debe tener las coordenadas fijas
-20010	*Genera_Orden*	La entidad Origen y Destino al llamar a esta función debe tener las coordenadas fijas
-20011	*Ejecutar_FinDeOrden*	Ha intentado finalizar una estación no definida en el sistema
-20012	*Ejecutar_FinDeOrden*	El transporte que se pretende finalizar no existe en la tabla

		transportes
-20013	*Ejecutar_FinDeOrden*	No existe ninguna trayectoria para la orden
-20014	*Ejecutar_FinDeOrden*	No existe trayectoria parcial para llegar a destino
-20015	*RESERVADO*	
-20016	*Ejecutar_FinDeOrden*	Las coordenadas de la nueva estación deben ser fijas y sin embargo aparecen como variables
-20017	*Ejecutar_FinDeOrden*	No se pueden encontrar los datos de la estación siguiente
-20018	*RESERVADO*	
-20019	*Buscar_Orden*	Procesar Orden ha sido llamado con una combinación Máquina/Tarjeta/Secuenciador no definida en Procesar_Ordenes
-20020	*Buscar_Orden*	El evento de búsqueda de orden definido ha devuelto un número de transporte que no existe en msg_tr
-20021	*RESERVADO*	
-20022	*RESERVADO*	
-20023	*Ejecutar_FinDeOrden*	La orden se pretende finalizar para otra estación diferente de la que aparece en la tabla transportes

5.4 El Agente de Transportes con PLC

En el caso de que se quiera usar el Sistema de Galileo para gobernar un sistema de control compuesto por PLCs, este actúa únicamente como el Agente de Transportes, es decir, realizando la tarea que antes llevaba a cabo el Monitor de Transportes. Desde el punto de vista de la programación, los PLCs son "cajas negras" para Galileo, y algo parecido ocurre a la inversa. Es necesario, por tanto, que exista un cierto protocolo de comunicación mediante el cual ambos sistemas puedan comunicarse. Esto se logra mediante la definición de unas estructuras a las que la parte PLC debe adherirse de forma obligatoria, y proporcionar su dirección a Galileo, a fin de que este sepa donde buscar esos datos. A partir de ahí, el Agente de Transporte de Galileo funcionara de la misma forma anteriormente descrita para las máquinas normales de Galileo, con la diferencia de que los datos de ordenes, fines de ordenes, PIEs y demás serán publicados en tiempo real en dichas estructuras, a fin de que la parte PLC pueda tener constancia de dichos cambios.

Dichas estructuras son de dos tipos: datos y control. Las estructuras de datos se refieren a los datos del transporte que las máquinas PLC mueven por la instalación, es decir, el equivalente a un tracking.

5.4.1 Estructura de tracking que Galileo comunica con un PLC

La estructura de un tracking que una máquina PLC debe observar, debe tener el siguiente formato:

Campo	Descripción
Numero de tracking	1 Entero sin signo de 32 bits
WTU	1 Entero sin signo, de 64 bits. Es el identificador de la UMA

Numero de entidad Origen	1 Byte sin signo
Tipo de entidad Origen	1 Byte sin signo
Posicion X de Origen	1 Entero sin signo de 16 bits
Posicion Y de Origen	1 Entero sin signo de 16 bits
Lado Origen	1 Byte sin signo
Pasillo Origen	1 Byte sin signo
Profundidad Origen	1 Byte sin signo
Tipo de UMA	1 Byte sin signo
Numero de entidad Destino	1 Byte sin signo
Tipo de Entidad Destino	1 Byte sin signo
Posicion X de Destino	1 Entero sin signo de 16 bits
Posicion Y de Destino	1 Entero sin signo de 16 bits
Lado Destino	1 Byte sin signo
Pasillo Destino	1 Byte sin signo
Profundidad Destino	1 Byte sin signo
Flag de Datos	1 Byte sin signo. RESERVADO
Datos Auxiliares	Se trata de un campo de datos, de tamaño 50 bytes.

5.4.1.1 Datos que Galileo usa para reconocer las averías de una máquina

Se necesita un campo de 16 bytes donde se van colocando las averías que la máquina va señalando. Los números de avería deben estar comprendidos entre 0 y 127, y se marcan activando el bit con ese número dentro del área de memoria, Es decir:

Byte= nºavería / 8
Bit = nº avería % 8

Por ejemplo:

Si la máquina quiere activar la avería número 1,
Byte ➔ 1 / 8 = 0
Bit ➔ 1%8 = 1
Es decir bit 1 del `byte` 0 de esa palabra se pondría a 1.
Si la avería a activar es la que tiene el código 100, entonces
Byte ➔ 100 / 8 = 12
Bit ➔ 100 %8 = 4
Es decir, el Bit 4 del `byte` 12 de ese área debería activarse.

5.4.2 Campos que son necesarios para que Galileo pueda trabajar con una maquina

Las estructuras de control, necesarias para realizar las búsquedas, fines de orden y PIE por parte de las máquinas PLC se describen a continuación.

5.4.2.1 Estructura de búsqueda

Campo	Descripción
`FindCommand`	1 Byte sin signo, indica que la máquina pide una orden.
`Found`	1 Byte sin signo, indica a la máquina que hay una

	orden lista para ella.
Posicion actual X	1 Entero sin signo de 32 bits
Posicion actual Y	1 Entero sin signo de 32 bits
Lado actual	1 Byte sin signo
Pasillo Actual	1 Byte sin signo
Estado actual	1 Byte sin signo. Indica el estado de la máquina.

5.4.2.2 Estructura para dar un PIE

Campo	Descripcion
NotifyIIS	1 Byte sin signo. Flag que indica que la máquina quiere dar un PIE.
AckIIS	1 Byte sin signo. Flag que indica a la máquina que el palet en PIE ha sido realizado
FlagsIIS	1 Entero sin signo. Diversos flags para indicar el PIE
Numero estacion PIE	1 Byte sin signo
Tipo estacion PIE	1 Byte sin signo
Datos Auxiliares PIE	Campo de datos de PIE, 256 bytes

5.4.2.3 Estructura para comunicar Fines de Orden

Campo	Descripción
EndCommand	1 Byte sin signo. Flag que indica que la máquina quiere finalizar un transporte
Ended	1 Byte sin signo. Le indica a la máquina que el transporte ha sido finalizado
EndTransport	1 Entero sin signo de 32 bits. Número de transporte a finalizar.
EndWTU	1 Entero de 64 bits. Identificador de la UMA del transporte que finaliza.
EndErrorCode	1 Entero sin signo de 32 bits. Código de error con el que finalizar la orden
Numero de estacion Fin de Orden	1 Byte sin signo
Tipo de estacion que finaliza la orden	1 Byte sin signo
Datos auxiliares para el fin de orden	1 Entero sin signo de 32 bits

6 SISTEMA SCADA: STATUS MONITOR

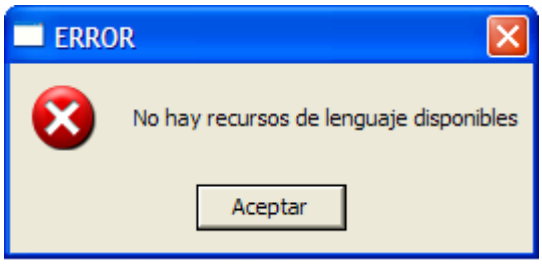
El Sistema de Control Galileo incorpora en su sistema de desarrollo Designer el módulo de visualización Status Master. Esta integración hace que la realización y diseño de la aplicación sea más sencilla e integrada. Se une a la potencia de un lenguaje como JavaScript ya utilizado en las versiones previas de Status Master toda la potencia del sistema de Objetos de la VCL (Visual Component Library) de Borland. La integración de todos estos elementos se ha conseguido mediante una exposición a través de interfaces COM de todos y cada uno de los objetos de la VCL y del API de Windows. De esta forma la capacidad de la visualización es infinita, no quedando ninguna tarea fuera de su alcance.

La técnica utilizada, similar a la utilizada por Microsoft en Internet Explorer y en el lenguaje de programación *Xana,* consiste en hacer interactuar al lenguaje interpretado con objetos binarios que residen en dlls de manera que se consigue un grado de efectividad mucho mayor mientras que se mantiene la flexibilidad de tener código interpretado y que por lo tanto no tiene que sufrir el largo proceso de compilación.

Todo lo referente a la configuración y lenguaje del Status Monitor aparece descrito en el manual del software de desarrollo Designer.

7 FAQS

Apartado dedicado a las incidencias más comunes que pueden surgir con el Sistema de Control Galileo.

PREGUNTA: **ERROR AL ABRIR LA CONSOLA DE GALILEO**
Al intentar arrancar la consola de Galileo desde el acceso directo del escritorio aparece el siguiente mensaje en pantalla:    ¿A qué puede ser debido?
RESPUESTA:
Este mensaje de error se genera cuando el acceso directo a la consola de Galileo del escritorio ha sido generado arrastrando directamente GalileoConsole.exe. Para poder generar correctamente dicho acceso a la consola, es necesario crear un acceso directo, cortar y pegarlo en el escritorio.

PREGUNTA: **GALILEO NO EJECUTA CÓDIGO**
Tengo un proyecto de Galileo el cual he compilado. Al arrancar el servicio de Galileo, detecto que desde el Designer en modo Online no aparece ninguna etapa activa. He revisado en la consola de Galileo en la pestaña de Máquinas y sucede lo mismo, no muestra ninguna etapa, como si no se ejecutase el código. Además en las pestaña de Imagen de Proceso tanto de entradas como de salidas no muestra nada. ¿Puede ser un problema con la versión de Galileo?
RESPUESTA:

El problema tiene que ver con el nombre del PC generalmente y no con la versión de Galileo, aunque es necesario revisar que sea la última publicada en el portal y que además coincida con la de Designer instalada en dicho PC.

El nombre del PC ha de ser IGUAL que el que tiene en la configuración de Windows y SIEMPRE EN MAYÚSCULAS, aunque en el nombre real del PC se encuentre en minúsculas.

PREGUNTA: **LA VISUALIZACIÓN NO ACTUALIZA LAS VARIABLES**

La visualización del proyecto parece que no actualiza ninguna variable de Galileo, sin embargo si que funcionan por ejemplo, la apertura de pantallas mediante botones

RESPUESTA:

Si una visualización no actualiza las variables de Galileo que consulta en cualquier elemento es que no se ha creado un componente TTimerTick. Este componente es el encargado de actualizar las variables de Galileo, por lo que es necesario que siempre exista. Además también es importante que solo exista un componente de este tipo, esté siempre habilitado y no tenga asociado ningún evento.

8 AVISO LEGAL