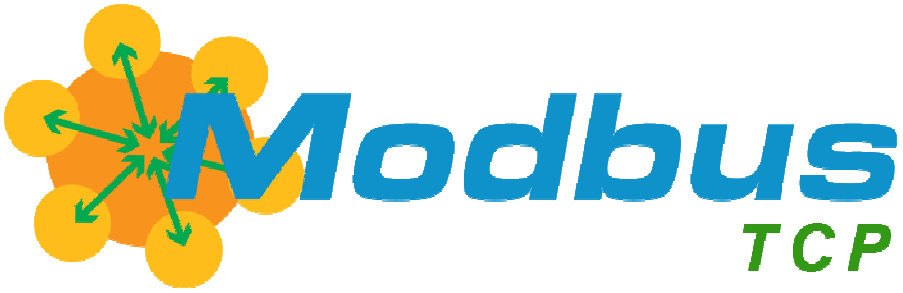
Slicetex Ladder Designer Studio

NOTA DE APLICACIÓN

AN021
ModBus TCP – Cliente (Master)

Autor: Ing. Boris Estudiez



[1]

Modelos Aplicables	AX, CX y DX

1 Descripción General

*La presente nota de aplicación explica como configurar y utilizar en modo cliente (Master) el protocolo **ModBus TCP** desde nuestros PLC.*

***ModBus** es un protocolo de comunicaciones creado originalmente por Modicon (ahora Schneider Electric) para su uso en PLC. Simple y robusto, el protocolo ModBus se convirtió en un protocolo estándar de facto con el paso del tiempo. Ampliamente difundido, ahora se utiliza para comunicar miles de dispositivos electrónicos industriales.*

***ModBus TCP** permite el empleo del protocolo original **ModBus** con dispositivos que se comunican mediante una red Ethernet (o cualquier otro medio físico) utilizando el stack TCP/IP (internet).*

*Este documento detalla el uso del protocolo **ModBus TCP** en modo cliente (Master), que le permitirá conectar el PLC a diferentes dispositivos **ModBus TCP** que funcionen como servidor (Slave) para enviar u obtener datos. Ejemplos completos se encuentran en nuestro sitio Web.*

[1]: Imagen propiedad de www.modbus.org.

2 Lecturas Recomendadas

Antes de leer este documento, recomendamos que se familiarice con el software StxLadder y el PLC adquirido. Sugerimos leer los siguientes documentos:

1. Manual de Usuario del software StxLadder.
2. Manual de Programación Pawn del PLC (si utiliza lenguaje Pawn)
3. Hoja de datos técnicos del PLC.

Mas documentación puede encontrar en la página del producto: www.slicetex.com.

Para consultas y soporte, ponemos a disposición un foro de discusión en: www.slicetex.com/foro donde puede leer preguntas de otros usuarios y realizar también sus propias preguntas.

Se recomienda leer el estándar del protocolo ModBus, disponible en www.modbus.org :

Protocolo ModBus:

http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

ModBus TCP:

http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf

2.1 Ejemplos

En nuestro sitio web, busque la pagina de la nota de aplicación AN021, desde dicha podrá encontrar ejemplos completos para utilizar en el PLC.

3 Requerimientos

Para esta nota de aplicación, debe tener instalado en su computadora el entorno de Programación **StxLadder** (Slicetex Ladder) y utilizar un firmware actualizado con soporte **ModBus** en el PLC.

Se recomienda estar familiarizado con los conceptos básicos del protocolo ModBus y su mecanismo.

4 Teoría de Funcionamiento

Los PLC de Slicetex Electronics permiten conectarse a un servidor **ModBus TCP** como cliente. La comunicación se realiza a través de una red Ethernet utilizando una comunicación TCP/IP (internet). Cabe destacar que existen routers o conversores que permiten conectar dispositivos ModBus desde redes Ethernet a redes RS-485.

Una transacción típica **ModBus TCP** se muestra en la Figura 1 a continuación:

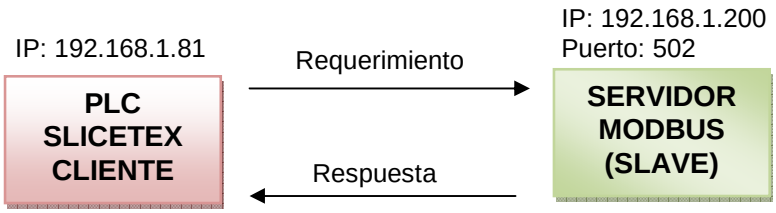


Fig. 1: Transacción ModBus TCP.

La figura 1, muestra el proceso que lleva a cabo el PLC cuando se realiza una transacción ModBus TCP para enviar u obtener datos de un servidor ModBus. Los pasos son los siguientes:

1. **Configuración**: El PLC debe configurar los parámetros para conectarse al servidor ModBus, como dirección IP y puerto de destino.
2. **Conexión**: El PLC establece una conexión TCP con el servidor, la cual estará abierta hasta que el cliente la cierre (por ejemplo para conectarse a otro servidor) o hasta que el servidor la cierre (por ejemplo porque el cliente está inactivo).
3. **Requerimiento**: El PLC envía una función ModBus al servidor para leer o escribir datos.
4. **Respuesta**: El servidor procesa el requerimiento del cliente, y devuelve una respuesta que depende de la función ModBus ejecutada.
5. **Recepción:** El PLC recibe la respuesta, la almacena en una memoria interna. El usuario a través de un programa Ladder o Pawn, puede leer la respuesta en busca de datos o determinar si existen errores.
6. **Fin:** El cliente puede realizar otro requerimiento al servidor o cerrar la conexión para desocupar al servidor.

En la implementación actual (a menos que otro documento detalle lo contrario), los PLC de Slicetex Electronics, pueden conectarse solo con un servidor a la vez. Si necesita obtener información de múltiples servidores, debe cerrar la conexión actual y luego conectarse al servidor deseado.

Tip: Para optimizar la velocidad de conexión, es recomendable no cerrar la conexión cada vez que se realizan requerimientos al Servidor. Pero tenga la precaución de mantener activa la misma, algunos servidores si no reciben requerimientos en un periodo de tiempo especifico (en general 30 o más segundos) pueden desconectar al cliente por inactividad.

4.1 Funciones ModBus Soportadas

En la tabla siguiente se listan las funciones ModBus que el PLC soporta como cliente, las cuales pueden ser utilizadas para hacer requerimientos en un servidor ModBus:

Tabla 1: Funciones ModBus Soportadas

Función	Código	Descripción
Read Coils	1	Leer estado de salidas discretas (en general el estado de los reles).
Read Discrete Inputs	2	Leer estado de entradas discretas.
Read Holding Register	3	Leer valores de registros “Holding”.
Read Input Register	4	Leer valores de registros de entrada de solo lectura.
Write Single Coil	5	Permite modificar el valor de una sola salida discreta.
Write Register	6	Escribe un valor en un registro.
Write Multiple Coils	15	Permite modificar el valor de múltiples salidas discretas al mismo tiempo.
Write Multiple Registers	16	Escribe múltiples registros al mismo tiempo.
Read / Write Multiple Registers	23	Escribe y lee múltiples registros al mismo tiempo.

4.2 Excepciones ModBus

Cuando un servidor ModBus detecta un error en la transacción, devuelve un error que corresponde a un Código de Excepción, los mismos se listan a continuación:

Tabla 2: Excepciones ModBus (Resumen)

Excepción	Código Hexa	Descripción
NONE	0	No hay error.
ILLEGAL FUNCTION	1	Función no soportada por el servidor.
ILLEGAL DATA ADDRESS	2	Dirección invalida.
ILLEGAL DATA VALUE	3	Valor de datos invalidos.
SLAVE DEVICE FAILURE	4	Error interno en el servidor.
ACNOWLEDGE	5	Función aceptada, pero se requiere un tiempo de procesamiento
SLAVE BUSY	6	El servidor no puede aceptar la petición, esta ocupado.
MEMORY PARITY ERROR	8	Falla en comprobación de memoria.
GATEWAY PATH FAILED	A	Ruta al Gateway no disponible.
GATEWAY TARGET FAILED	B	El dispositivo remoto fallo en respuesta, el Gateway genera este error.

5 ModBus Cliente con Lenguaje Ladder

En esta sección explicaremos a modo general como utilizar el protocolo ModBus en modo cliente con el lenguaje Ladder.

Puede bajar ejemplos completos de esta nota aplicación en nuestro sitio Web.

En este documento llamaremos “transacción” al proceso de enviar un requerimiento y obtener respuesta desde el servidor ModBus.

Llamaremos “conexión” al periodo de tiempo que la conexión entre el cliente y el servidor esta activa.

En lenguaje Ladder es muy simple conectarse al servidor ModBus TCP. Básicamente hay cuatro clases de componentes disponibles:

- Componentes para configurar el cliente y la conexión.
- Componentes para enviar transacciones al servidor.
- Componentes para leer respuesta del servidor y/o errores en la transacción.
- Componentes para activar y desactivar eventos.

En la página siguiente repasaremos brevemente como utilizar **ModBus TCP** como cliente en Ladder.

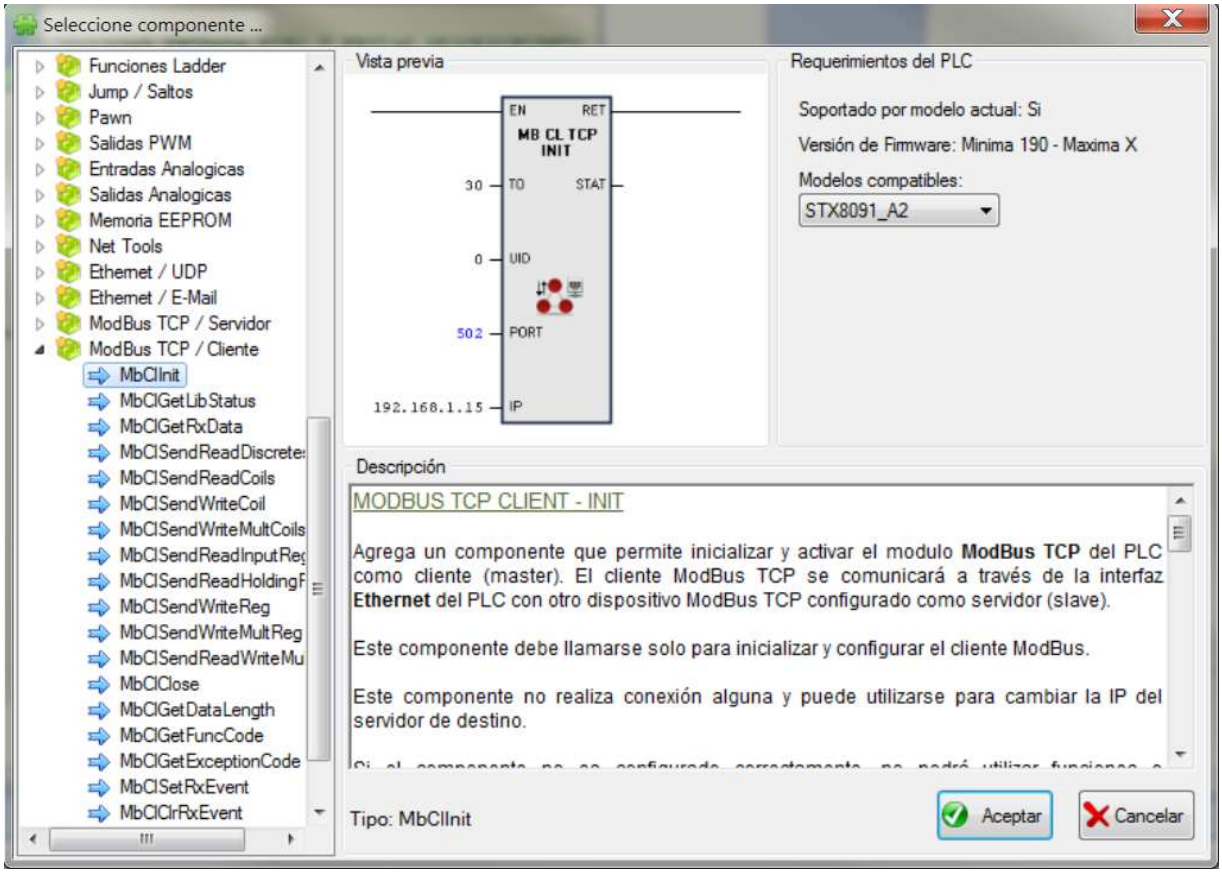
Importante:

Recuerde que la documentación detallada de cada componente está disponible en el mismo entorno StxLadder. Para acceder a dicha documentación, solo tiene que insertar el componente, luego seleccionarlo con el botón-derecho del mouse, y posteriormente acceder al ítem “**Ver descripción del componente …**” en el menú contextual desplegable del componente.

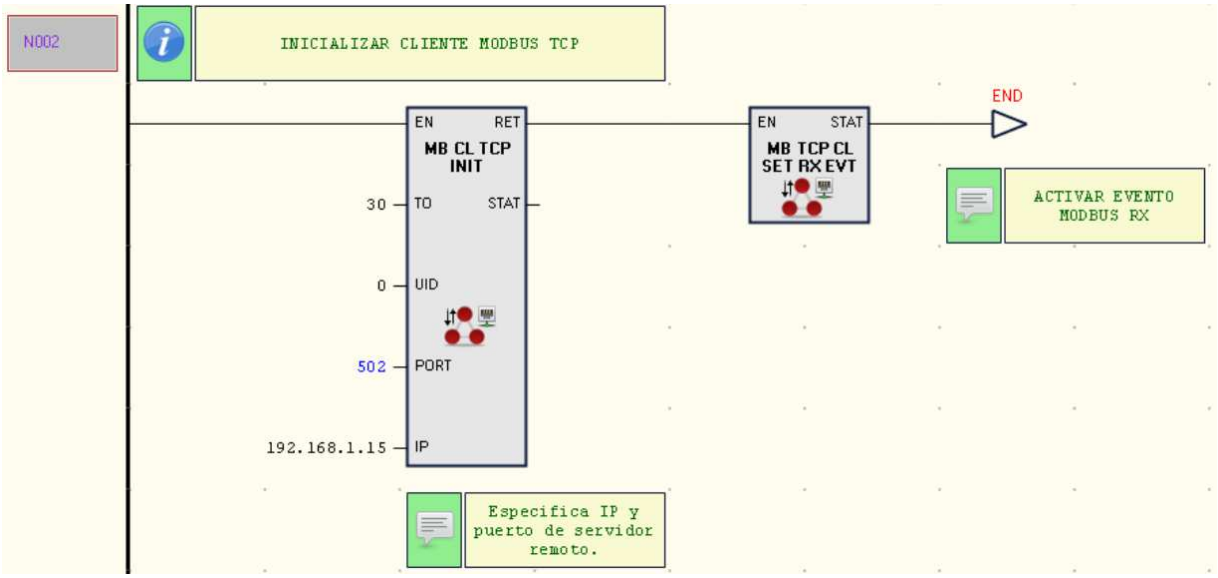
5.1 Componentes Ladder

5.1.1 Configuración

Los componentes Ladder para el cliente ModBus TCP, se encuentran en el selector de componentes del entorno StxLadder. Dentro del selector, navegue dentro de la sección "**ModBus TCP / Cliente**". Luego puede seleccionar el componente **MbClInit**, como se muestra en la imagen a continuación:



Puede insertar el componente **MbClInit** en el diagrama **Inicio.slp** como se muestra a continuación:



La descripción completa del componente, puede encontrarse en el entorno StxLadder, sin embargo, describiremos a continuación brevemente el mismo:

La entrada **EN** ejecuta el componente si el flujo de corriente es "1". La salida del componente **RET** es "1" si el mismo puedo ser ejecutado con éxito. En caso de error, puede obtener un código de error en la salida **STAT** del componente, que le proporcionará más datos del tipo de error.

La entrada **TO**, específica el tiempo en segundos que debe esperar el cliente a una respuesta de un requerimiento. Es decir, si especificamos 30 segundos por ejemplo, luego de enviar un requerimiento al servidor, nuestro cliente esperará al menos 30 segundos por una respuesta antes de finalizar con código de error (timeout).

El puerto "**UID**" especifica el Unit Identifier. Se utiliza para direccionar dispositivos conectados a través de un Gateway ModBus. Si no se utiliza, usar valor 0.

La entrada **PORT**, específica el número de puerto TCP donde el servidor ModBus TCP está escuchando conexiones de clientes remotos. Es el puerto TCP donde intentará conectarse nuestro cliente. Se utiliza en general 502.

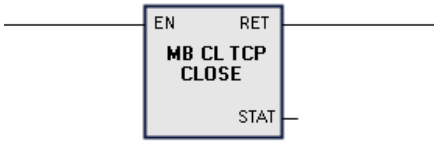
En la entrada "**IP**" se especifica la dirección internet del servidor ModBus TCP de destino. Puede ser constante o una variable tipo **Int32_Array** de 4 elementos (primer elemento corresponde al octeto más significativo). En este caso, definimos que el servidor se encuentra en la IP: 192.168.1.15.

Si el componente es ejecutado sin errores, el cliente ModBus TCP está listo comenzar a realizar transacciones ModBus.

Conectado en serie al componente de inicialización, pusimos otro componente llamado "**MbClSetRxEvent**", el cual nos permite activar el evento **OnMbClientRx**. Este evento se ejecuta al finalizar una transacción ModBus. Es muy útil para obtener los datos recibidos o el tipo de error en caso de falla.

Componente MbClClose:

Cierra una conexión establecida con el servidor **ModBus TCP**. Si la conexión no fue establecida, la función no tiene efecto y devuelve un código de error.



La salida "**RET**" del componente es verdadera si la conexión fue cerrada con éxito.

La salida "**STAT**" del componente, es de uso opcional, pero permite determinar si el **cliente** pudo cerrar la conexión. Es posible especificar una variable que almacenará un código de error. Si el retorno es cero, la operación es exitosa, si es negativo se produjo un error de activación.

- Retorno 0: Operación exitosa.
- Retorno -1: Error, no hay conexión establecida con el servidor.
- Retorno -2: Error, no hay conexión asignada, número máximo de conexiones alcanzado.
- Retorno -4: Error, la librería no fue inicializada.

Importante:

Este componente utiliza para cerrar una conexión con un servidor ModBus, informándole explícitamente que el cliente se desconectó. Se recomienda utilizar este componente al cambiar de servidor (dirección IP) o cuando no utilizamos más la conexión.

Si se envió un requerimiento al servidor ModBus, pero este no existe porque está apagado (por ejemplo), utilizar esta función no cierra la conexión, ya que la misma no existe y devuelve error. El cliente esperará un tiempo (55 segundos aproximadamente) y luego informará que existe un timeout.

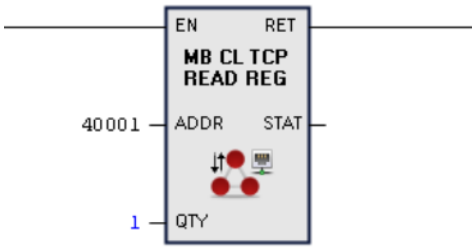
5.1.2 Lectura / Escritura de Registros

Los siguientes componentes permiten leer y escribir registros de 16-bits del servidor ModBus TCP.

Nota: Recuerde que los componentes envían el tipo de función ModBus mediante una conexión TCP. Luego de ejecutado el componente, comienza la conexión al servidor. Solo cuando el servidor responde, se obtiene la respuesta. Puede usar el evento **OnMbClientRx** para procesar la respuesta una vez recibida, sin necesidad de comprobar constantemente el estado de la transacción.

Componente MbClSendReadInputReg:

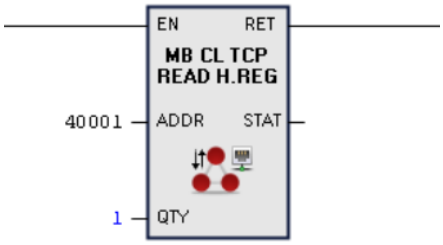
Permite leer Input Registers del servidor.



La entrada **ADDR** especifica la dirección del primer registro a leer y en **QTY** la cantidad de registros a leer.

Componente MbClSendReadHoldingReg:

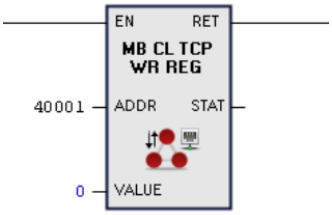
Permite leer Holding Registers del servidor.



La entrada **ADDR** especifica la dirección del primer registro a leer y en **QTY** la cantidad de registros a leer.

Componente MbClSendWriteReg:

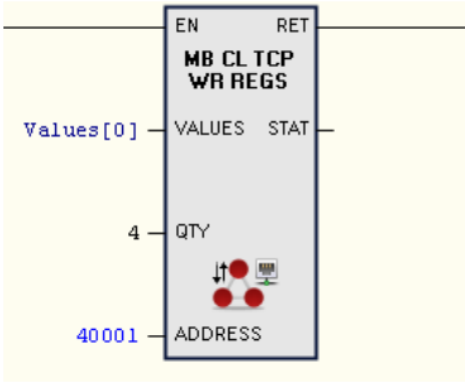
Permite escribir un único registro en el servidor.



La entrada **ADDR** especifica la dirección del registro a escribir y en **VALUE** el valor propiamente dicho a escribir.

Componente MbClSendWriteMultReg:

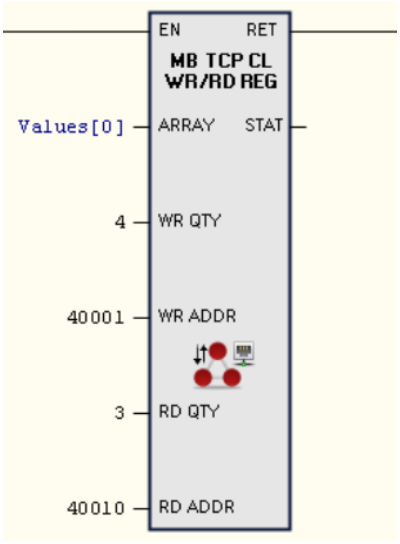
Permite escribir múltiples registros en el servidor.



La entrada **VALUES** admite un array de elemento tipo Int32_Array. Cada elemento representa el valor a escribir en cada registro. Por ej: en este caso el el elemento Values[0] se escribirá en la dirección 40001, el elemento Values[1] en la dirección 40002, y así sucesivamente hasta completar los 4 registros a escribir según el valor de la entrada **QTY**. Notar que la dirección del primer registro a escribir lo determina la entrada **ADDRESS**.

Componente MbClSendReadWriteMultReg:

Permite escribir y leer múltiples registros en el servidor al mismo tiempo.



Este componente es una combinación, entre los componentes para leer registros y los componentes para escribir registros. La ventaja radica en que todo se realiza en la misma conexión.

La entrada **ARRAY** permite especificar un array de valores, donde cada elemento del array representa un registro a escribir. La cantidad de registros a escribir la determina **WR QTY** y la dirección de escritura se define en **WR ADDR**.

Así mismo, el puerto **RD QTY** determina la cantidad de registros a leer a partir de la dirección definida en el puerto **RD ADDR**.

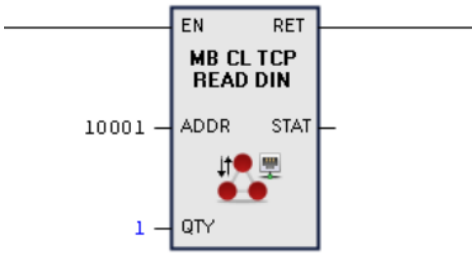
5.1.3 Lectura / Escritura de Bits

Los siguientes componentes permiten leer y escribir unidades de memoria de 1-bit.

Nota: Recuerde que los componentes envían el tipo de función ModBus mediante una conexión TCP. Luego de ejecutado el componente, comienza la conexión al servidor. Solo cuando el servidor responde, se obtiene la respuesta. Puede usar el evento **OnMbClientRx** para procesar la respuesta una vez recibida, sin necesidad de comprobar constantemente el estado de la transacción.

Componente MbClSendReadDiscretes:

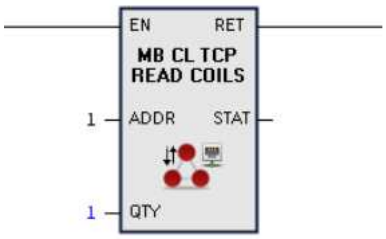
Permite leer una entrada discreta/digital o bit de memoria del servidor.



En la entrada "**ADDR**" se especifica una constante con la dirección de la primera entrada a leer y en entrada "**QTY**" especifica la cantidad de entradas a leer.

Componente MbClSendReadCoils:

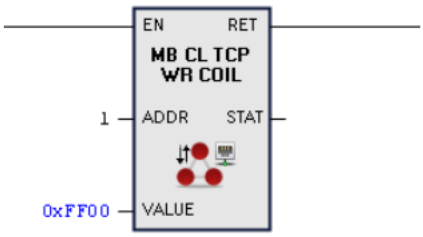
Permite leer una salida discreta/digital o bit de memoria del servidor.



En la entrada "**ADDR**" se especifica una constante con la dirección de la primera salida a leer y en entrada "**QTY**" especifica la cantidad de salidas a leer.

Componente MbClSendWriteCoils:

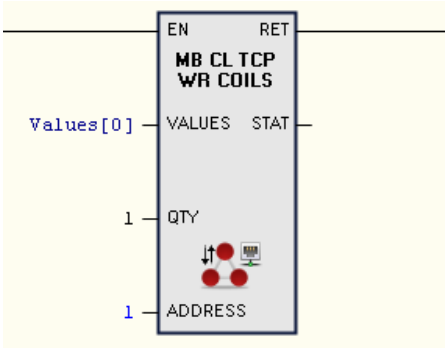
Permite escribir una salida discreta/digital o bit de memoria del servidor.



En la entrada "**ADDR**" se especifica una constante con la dirección de la salida a escribir. La entrada "**VALUE**" especifica el valor a escribir. El valor 0 desactiva la salida y el valor FF00 (hexadecimal) activa la salida.

Componente MbClSendWriteMultCoils:

Permite escribir una salida discreta/digital o bit de memoria del servidor.



La entrada "**VALUES**" se especifica un array del tipo **Int32_Array** que contiene los valores a escribir. De cada elemento del array solo se toman los 8 bits menos significativos, es decir solo 1 byte por elemento. Cada bit del elemento representa el valor de una coil. El array debe tener **QTY** / 8 elementos (sumar 1 si la división da menor a 1). El tamaño se comprueba automáticamente al compilar el proyecto.

La entrada "**QTY**" especifica la cantidad de salidas a escribir.

En la entrada "**ADDRESS**" se especifica una constante con la dirección de la primera salida a escribir.

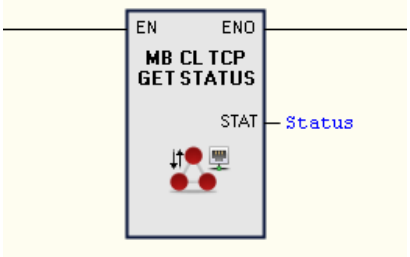
5.1.4 Lectura de Respuesta y de Errores

Los siguientes componentes permiten leer una respuesta recibida del servidor ModBus TCP. Tambien permiten leer códigos de estado de la librería y códigos de error retornados.

Es natural utilizar estos componentes en el evento **OnMbClientRx**, ya que el mismo es activado al finalizar una transacción, lugar donde puede procesar la respuesta.

Componente MbClGetLibStatus:

Permite obtener el estado de la librería **ModBus TCP Cliente** del PLC.



Este componente debe utilizarse para determinar el estado de una transacción **ModBus TCP** o identificar errores.

Siempre debe comprobar el código de estado antes de realizar una operación. Un código de estado negativo, implica una condición de error. Un código positivo implica un estado particular. El valor cero significa que el cliente recibió respuesta del servidor.

Cuando realiza una transacción **ModBus TCP**, el PLC comienza a negociar con el servidor **ModBus TCP** la transmisión del comando y la respuesta del mismo. Este proceso puede tomar desde decenas de milisegundos, hasta 125 segundos (o más) dependiendo del tiempo de respuesta de los servidores de internet.

Para evitar bloquear el PLC hasta que la respuesta del comando sea recibida (proceso que tarda decenas de segundos), puede utilizar este componente para leer el estado de la transacción. Usted puede leer el estado de transmisión cada X tiempo y luego setear una variable que indique su recepción. Bajo este escenario, es útil el componente **MbClGetLibStatus**.

Entradas:

- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.

Salidas:

- La salida "**ENO**" del componente es una copia del valor de la entrada "**EN**".
- La salida "**STAT**" del componente devuelve en una variable tipo **Int32** el estado de la librería. Un código de estado negativo, implica una condición de error. Un código positivo implica un estado particular. El valor cero significa que el cliente recibió respuesta del servidor **ModBus TCP**.

Tabla 3: Códigos de estados retornado por MbClGetLibStatus en puerto STAT

Nombre	Valor	Descripción
MBCL_STAT_TRY_TO_SEND	127	Requerimiento en proceso de envío.
MBCL_STAT_LIB_NOT_INIT	126	La librería no fue inicializada aun.
MBCL_STAT_RESP_RXED	125	Respuesta recibida, pero no procesada.
MBCL_STAT_INITIALIZED	124	La librería fue inicializada.
MBCL_STAT_CLOSED	123	La conexión fue cerrada por el cliente.
MBCL_STAT_OK	0	Respuesta recibida con éxito del servidor, listo.
MBCL_STAT_ERR_ABORTED	-1	Error, conexión abortada.
MBCL_STAT_ERR_CONN_TO	-2	Error, timeout en conexión. Usualmente cuando el servidor no existe.
MBCL_STAT_ERR_CLOSED	-3	Error, conexión cerrada por el host.
MBCL_STAT_ERR_TO	-4	Error, timeout en transacción. Especificado por función de inicialización.
MBCL_STAT_ERR_ALLOC	-5	Error, la conexión no fue asignada.
MBCL_STAT_ERR_FUNC	-6	Error, ver código de retorno de función.
MBCL_STAT_ERR_EXCEP	-7	Error, se recibió un código ModBus de excepción.
MBCL_STAT_ERR_BAD_TID	-8	Error, TID (transaction ID) recibido difiere.
MBCL_STAT_ERR_BAD_UID	-9	Error, UID (unit ID) recibido difiere.
MBCL_STAT_ERR_FUNC_DIF	-10	Error, código de función ModBus recibido difiere al enviado.
MBCL_STAT_ERR_BAD_PID	-11	Error, PID (protocol ID) recibido incorrecto.

Importante:

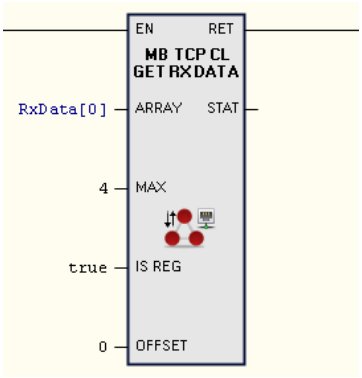
- Recuerde inicializar la librería con el componente **MbClInit** para evitar el código “126”.

Notas:

- Los códigos más importantes son el 127 (cuando hay una transacción en curso) y el 0 (cuando la respuesta del servidor está disponible). Para comprobar una situación de error, solo hay que verificar si el código devuelto es negativo.

Componente MbClGetRxData:

Obtiene los datos de la última respuesta recibida del servidor Modbus.



Entradas:

- El componente se ejecuta cuando el valor del flujo de corriente en el puerto de entrada "**EN**" es 1.
- La entrada "**ARRAY**" se especifica un array del tipo **Int32_Array** donde se copiarán los datos recibidos.
- La entrada "**MAX**" especifica el número máximo de bytes / registros a copiar. Máximo (32 o 16).
- En la entrada "**IS REG**" se especifica **true** si los datos recibidos contienen registros de 16-bits. Utilizar **falsen** si los datos recibidos son bytes de 8-bits.
- La entrada "**OFFSET**" especifica el número o desplazamiento en bytes a partir donde se comienzan a leer los registros de la respuesta ModBus. Muy útil cuando al comienzo de la trama hay información de la respuesta, pero no son propiamente los registros de 16-bits devueltos.

Salidas:

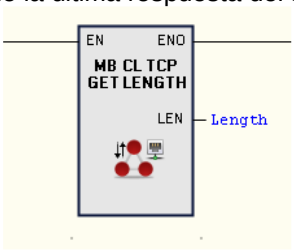
- La salida "**RET**" del componente es verdadera o "**1**" si los datos se copiaron con éxito. Si es "**0**", existe un error. Puede encontrar más información del error en el puerto "**STAT**".
- La salida "**STAT**" del componente, es de uso opcional, pero permite determinar una condición de error. Es posible especificar una variable que almacenará un código. Si el retorno es cero, la operación es exitosa, si es negativo se produjo un error.
    - Retorno 0: Operación exitosa, datos copiados.
    - Retorno -1: Error, no hay datos disponibles para copiar.
    - Retorno -2: Error, numero incorrecto de datos a copiar.
    - Retorno -4: Error, dirección del array inválida.

Consejos:

- Antes de leer en buffer, puede ser útil verificar el estado de la librería con el componente **MbClGetLibstatus**.
- Puede leer la respuesta del requerimiento del servidor desde el evento **OnMbClientRx**.

Componente MbClGetDataLength:

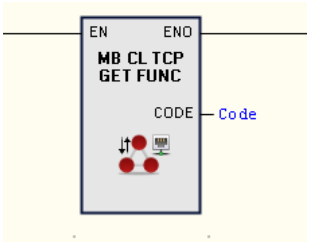
Obtiene la cantidad de bytes recibidos de la última respuesta del servidor Modbus.



La salida "**LEN**" del componente devuelve la cantidad de bytes recibidos, numero positivo mayor o igual a 0. Si el valor es negativo, implica un error en la librería o de inicialización.

Componente MbClGetFuncCode:

Obtiene el código de función ModBus de la última respuesta recibida del servidor Modbus.

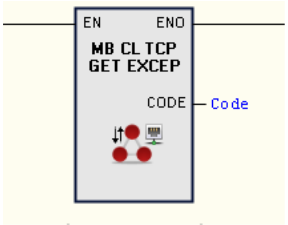


La salida "**CODE**" del componente devuelve el código del la última función ModBus recibida, numero positivo mayor o igual a 0 (ver tabla a continuación). Si el valor es negativo, implica un error en la librería o de inicialización.

Ver **Tabla 1** en página 4 para valores de códigos de funciones Modbus.

Componente MbClGetExceptionCode:

Obtiene el código de excepción ModBus de la última respuesta recibida del servidor Modbus.



La salida "**CODE**" del componente devuelve el código del la última excepción ModBus recibida, numero positivo mayor o igual a 0 (ver tabla a continuación). Si el valor es negativo, implica un error en la librería o de inicialización.

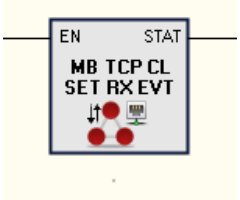
Ver **Tabla 2** en página 4 para valores de códigos de excepciones Modbus.

5.1.5 Control de Eventos

Los siguientes componentes permiten controlar los eventos disponibles del cliente ModBus TCP.

Componente MbClGetFuncCode:

Permite activar el evento "**OnMbClientRx**".



El evento "**OnMbClientRx**" se activa cuando una transacción termina y el código de estado de la librería (ver componente **MbClGetLibStatus** para constantes) es igual o menor a 0.

Al dispararse el evento, el PLC llama al diagrama asignado para manejar el evento. Dicho evento se define desde el **Explorador de Proyecto** del entorno **StxLadder**. El programador procesará el evento con la lógica especificada en el diagrama.

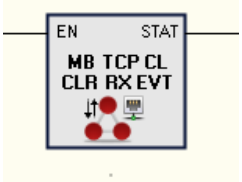
El evento **OnMbClientRx** le permite utilizar de forma asíncrona los componentes para realizar requerimientos al servidor ModBus.

La salida "**STAT**" del componente es verdadera si el evento pudo configurarse con éxito.

Nota: Recuerde crear el diagrama para procesar el evento.

Componente MbClSetRxEvent:

Permite desactivar el evento "**OnMbClientRx**".



La salida "**STAT**" del componente es verdadera si el evento pudo desactivarse con éxito.

5.2 Ejemplo Ladder

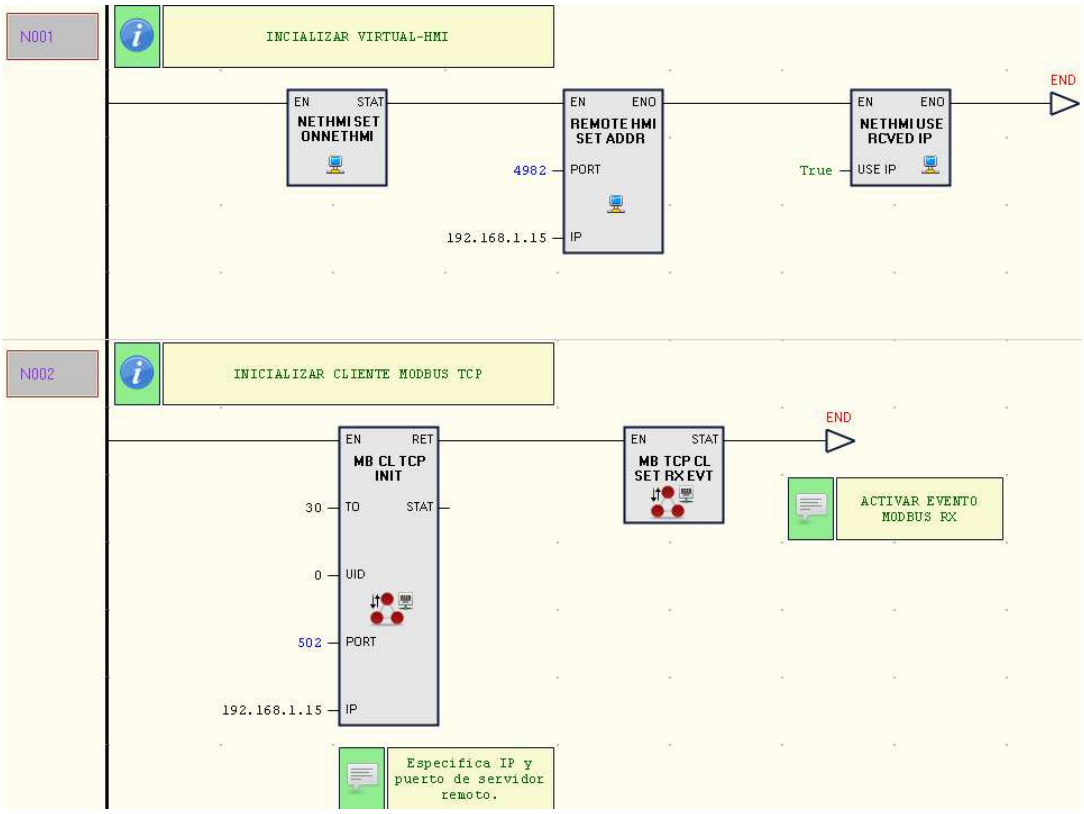
El siguiente ejemplo Ladder es un programa que realiza una conexión ModBus TCP a un servidor ModBus TCP y envía el comando modbus “Read Holding Register” cuando la entrada digital DIN1 del PLC es “1”. El comando “Read Holding Register” envía una petición para leer 4 registros de 16-bits a partir de la dirección 0.

A su vez, imprime mensajes en VirtualHMI para informar el estado de la transacción e imprimir los registros retornados por el servidor.

Este ejemplo puede bajarse desde nuestra página web, bajo el nombre “**MbTcpClientLadderEvn**”.

5.2.1 Diseño

Diagrama **Inicio.sld**:

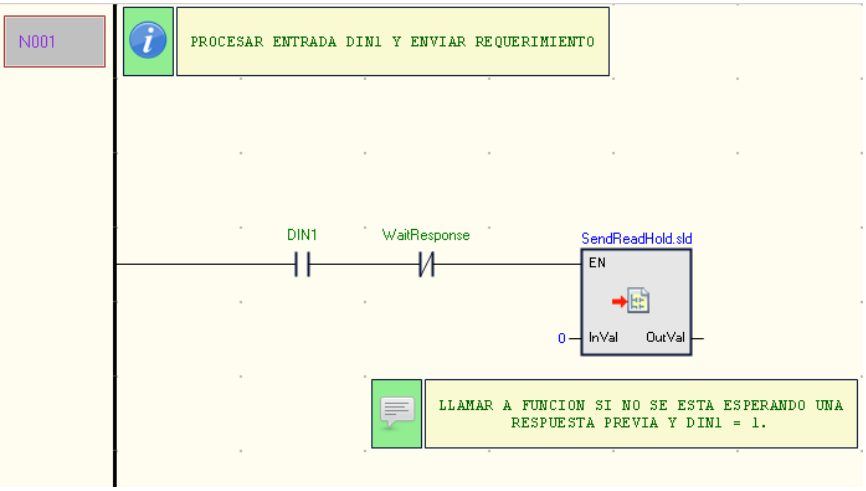


La network **N001** inicializa y configura los parámetros para el terminal VirtualHMI.

La **N002** inicializa el cliente ModBus TCP. Se especifica IP y puerto remoto del servidor ModBus TCP al cual vamos a conectarnos. En este caso 192.168.1.15. El timeout se establece en 30 segundos.

También se activa el evento “**OnMbClientRx**” con el componente **MbClSetRxEvent**.

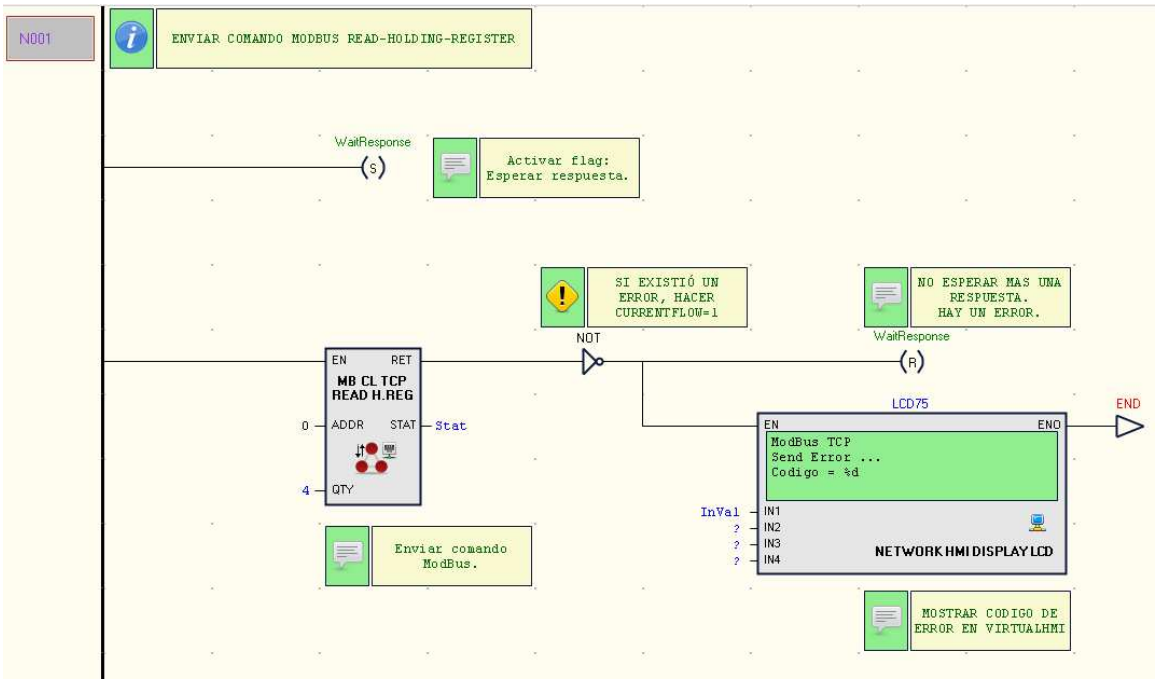
Diagrama **Principal.sld**:



La **N001**, constantemente lee la entrada DIN1. Si DIN1 vale 1 y la variable booleana **WaitResponse** es 0, se ejecuta el diagrama "SendReadHold.sld". En dicho diagrama se iniciará la transacción ModBus TCP.

La variable **WaitResponse** es tipo **Bool** y global. Cuando es "1" significa que el cliente comenzó una transacción ModBus TCP y está esperando una respuesta.

Diagrama **SendReadHold.sld**:



El diagrama "**SendReadHold.sld**" es una función-ladder definida por el usuario. En la misma, se inicia la transacción ModBus TCP cuando es llamada desde el diagrama **Principal.sld**.

Primero se setea la variable **WaitResponse** a 1, ya que vamos a comenzar una transacción que espera una respuesta por parte del servidor.

Luego llamamos al componente **MbClSendReadHoldingReg** que envía un requerimiento modbus del tipo "Read Holding Registers". En este caso utilizamos una dirección de registros "0" y pedimos leer una cantidad de 4 registros.

La salida **RET** del componente es "1" si el mismo fue enviado con éxito.

Caso contrario, la salida es **RET=0**. Esto implica que el flujo de corriente sea "0" también. Por lo tanto usamos un componente **NOT** para invertir el flujo de corriente en caso de error y resetear a 0 la variable **WaitResponse** e imprimir el código de error en VirtualHMI como muestra la siguiente figura:

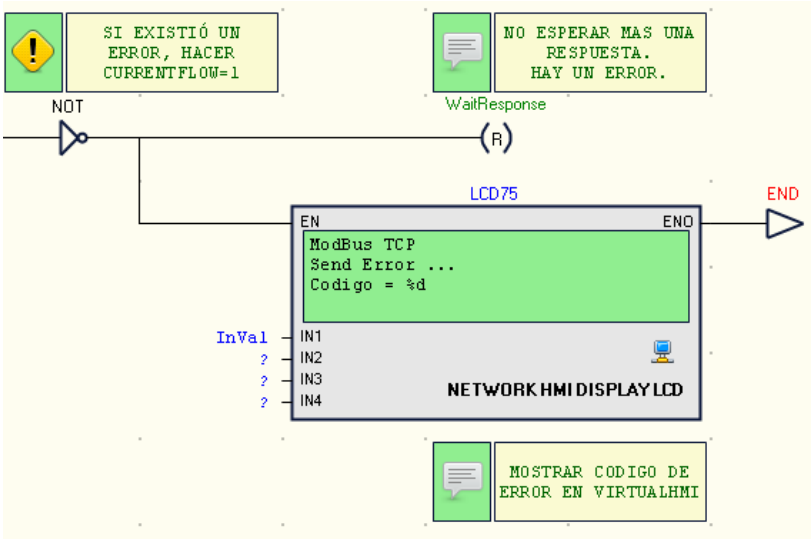
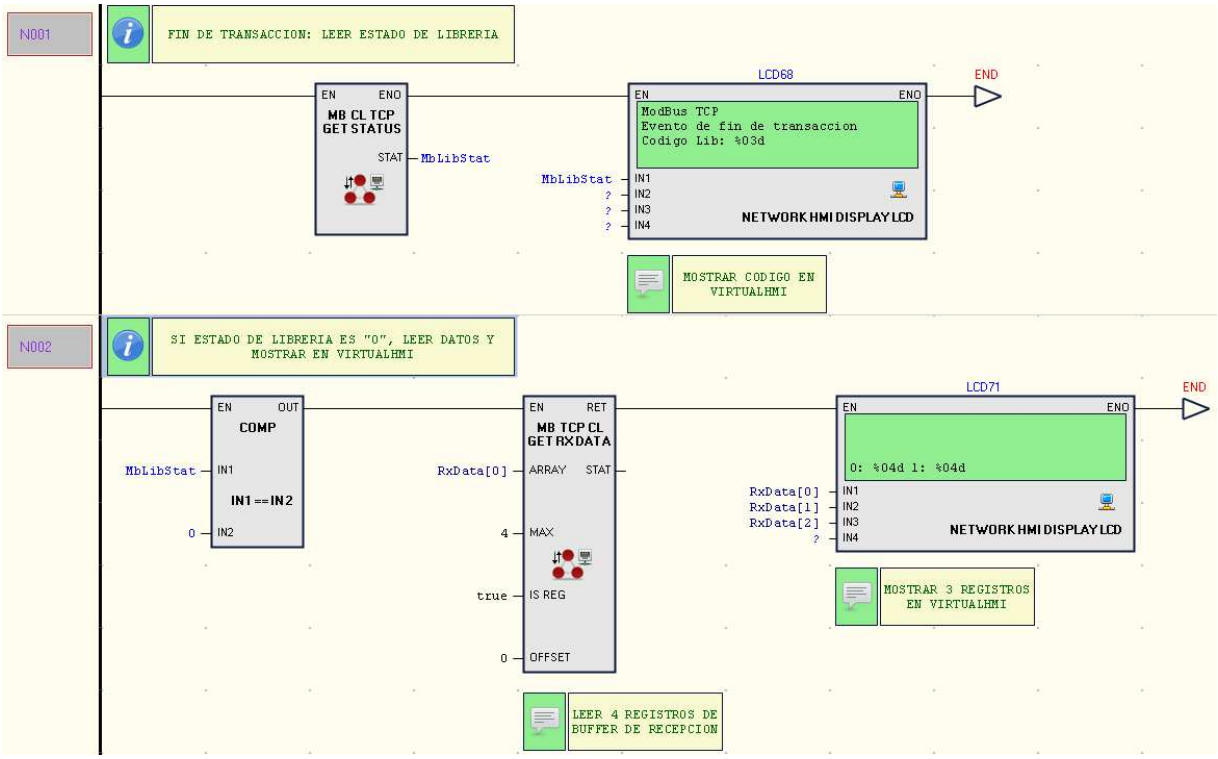


Diagrama **OnMbClientRx.sld**:



Este diagrama corresponde al evento **OnMbClientRx**, y se ejecuta el finalizar una transacción ModBus (ya sea por una respuesta recibida o un error).

Primero, en la **N001** obtenemos el código de estado de la librería ModBus TCP cliente con el componente **MbClGetLibStatus**. El código será almacenado en la variable **MbLibStat** y luego impreso en VirtualHMI.

Luego en la **N002**, comparamos el valor de la variable **MbLibStat** obtenido. Si es “0”, significa que tenemos una respuesta válida del servidor. Por lo tanto procedemos a llamar al componente **MbClGetRxData**.

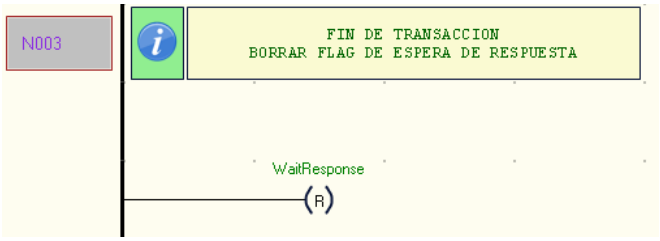
En el componente **MbClGetRxData** pasamos el array RxData[], donde copiaremos como máximo 4 registros recibidos (cada uno de 16-bits). Notemos que la entrada “**IS REG**” del componente es “true”, significando que los datos a copiar son registros y no simple bytes de 8-bits.

En la entrada “**OFFSET**” del componente **MbClGetRxData** indicamos **0-byte** de desplazamiento.

Sobre el array **RxData[]**, en los elementos **RxData[0]**, **RxData[1]**, **RxData[2]** y **RxData[3]** se almacenarán los 4 registros leídos.

En VirtualHMI solo los primeros tres registros serán impresos según especificamos en el componente **NetHmiLcdPrintf** conectado en serie con **MbClGetRxData**.

Finalmente en **N003**, luego de leer e imprimir los registros recibidos hacemos la variable **WaitResponse**= 0, es decir finalizamos la espera de respuesta.



5.2.2 Prueba del Ejemplo

Compile el proyecto seleccionando adecuadamente su modelo de PLC.

Transfiera el programa al PLC.

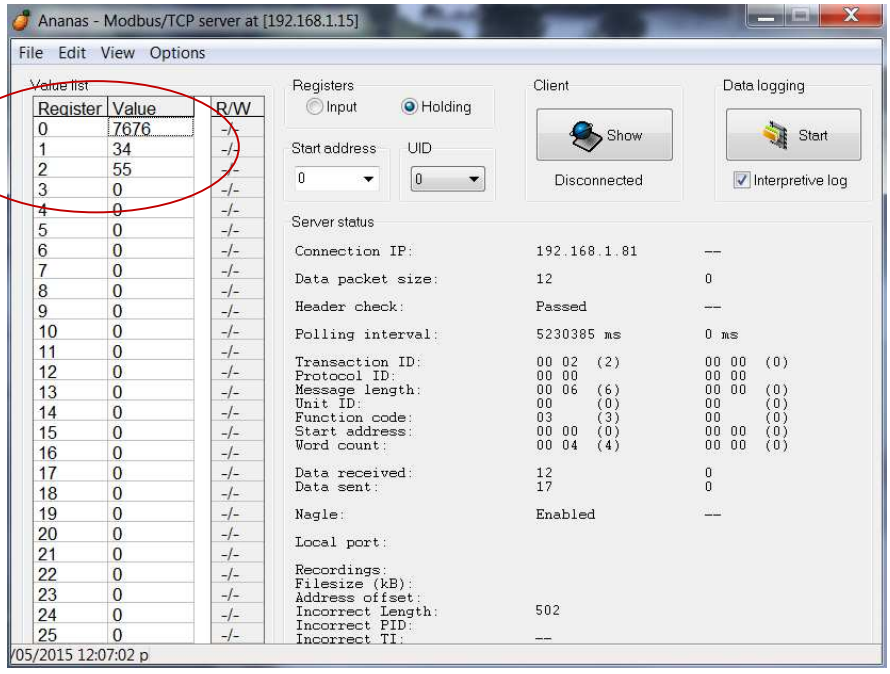
Ejecute el programa VirtualHMI, haga click en el boton "**ON**" de las teclas "**Action Key**".
Esto le permitirá al PLC identificar la IP de **VirtualHMI** para enviar mensajes de impresión en LCD.

Alternativa 1:

Puede probar el cliente ModBus TCP del PLC, con el servidor "**Ananas ModBus TCP Server para Windows**", visite nuestro sitio web para descargarlo.

http://www.slicetex.com/an/an021

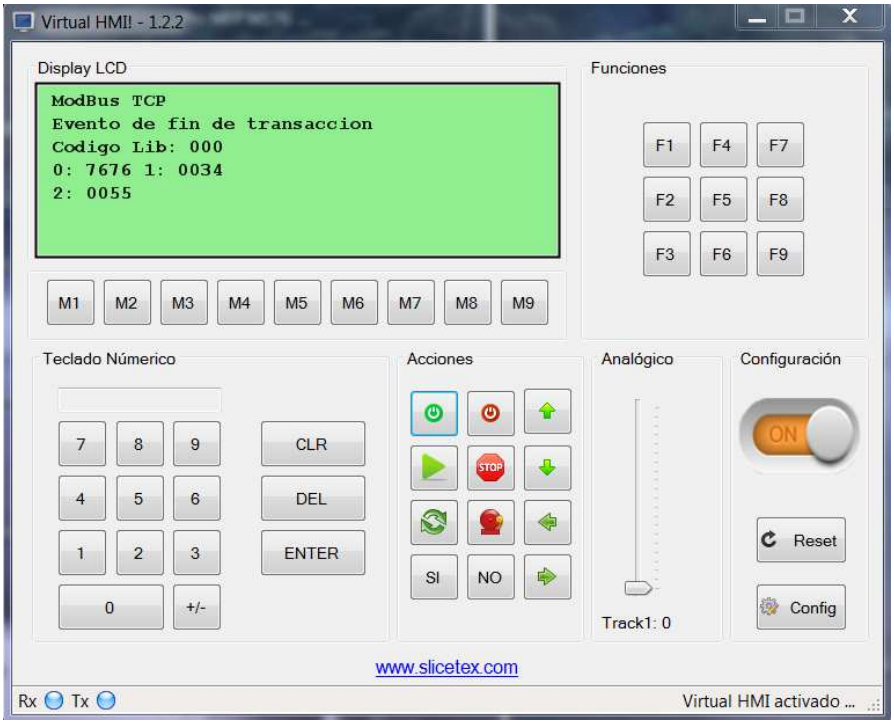
Valores de registros Holding (especificados por el usuario)



Ananas ModBus TCP Server – Corriendo en IP: 192.168.1.15

Una vez ejecutado el servidor ModBus TCP en su PC, debe especificar la dirección IP al cliente del PLC y luego puede probar enviando requerimientos con la entrada discreta DIN1.

Visualice en VirtualHMI la respuesta del comando ModBus enviado.



Respuesta en VirtualHMI
Note los valores de registros recibidos coinciden con Ananas Server

Alternativa 2:

Puede utilizar otro PLC de Slicetex o dispositivo como servidor ModBus TCP y realizar el requerimiento ModBus para leer los Holding Registers. Recuerde especificar un IP y dirección de registros validas.

6 ModBus Cliente con Lenguaje Pawn

En esta sección explicaremos a modo general como utilizar el protocolo ModBus en modo cliente con el lenguaje Pawn.

Puede bajar ejemplos completos de esta nota aplicación en nuestro sitio Web.

En este documento llamaremos “transacción” al proceso de enviar un requerimiento y obtener respuesta desde el servidor ModBus.

Llamaremos “conexión” al periodo de tiempo que la conexión entre el cliente y el servidor esta activa.

En lenguaje Pawn es muy simple conectarse al servidor ModBus TCP. Básicamente hay cuatro clases de funciones disponibles:

- Funciones para configurar el cliente y la conexión.
- Funciones para enviar transacciones al servidor.
- Funciones para leer respuesta del servidor y/o errores en la transacción.
- Funciones para activar y desactivar eventos.

En la página siguiente se describen dichas funciones en detalle.

6.1 Funciones Nativas en Pawn Disponibles

6.1.1 Funciones de Configuración

MbClInit(ServerIP, ServerPort, Timeout, UID): Inicializa y configura los parámetros para que el PLC pueda conectarse a un servidor ModBus TCP cuando realice una transacción. Esta función no realiza conexión alguna y puede utilizarse para cambiar la IP del servidor de destino.		
Argumentos	**Tipo**	**Descripción**
ServerIP	E	Representación de 32-bit de la dirección IP del servidor ModBus TCP.
ServerPort	E	Puerto TCP del servidor ModBus TCP, en general se utiliza el 502.
Timeout	E	Tiempo en segundos que debe esperar el cliente a una respuesta de un requerimiento. Mínimo 1 y Máximo 125.
UID	E	Unit Identifier. Se utiliza para direccionar dispositivos conectados a través de un Gateway modbus. Si no se utiliza, usar valor 0.
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, falla en inicialización, memoria insuficiente.
-10	S	Error, el cliente ModBus ya está conectado al servidor, desconéctese primero o espere desconexión antes de continuar.
Notas		**Descripción**
1		Esta función no realiza conexión al servidor ni envía datos por la red.

Ejemplo 1:

```
// Almacenar en la siguiente variable la dirección IP del servidor:
new ServerIp = IpToVar(192,168,1,15)

// Inicializar Cliente ModBus TCP.
// Conectar a la dirección IP 192.168.1.15 puerto 502.
// Utilizar 30 segundos de timeout para respuesta de transacción.

MbClInit(ServerIp, 502, 30, 0)
```

Ejemplo 2:

```
// Inicializar Cliente ModBus TCP.
// Conectar a la dirección IP 192.168.1.15 puerto 502.
// Utilizar 30 segundos de timeout para respuesta de transacción.

if( MbClInit(IpToVar(192,168,1,15), 502, 30, 0) < 0 )
{
      // Error!.
}
```

6.1.2 Funciones de Transacción

Las siguientes funciones enviar un requerimiento al Servidor ModBus. El usuario debería comprobar si el requerimiento fue enviado exitosamente y luego esperar la respuesta.

MbClSendReadCoils(StartAddr, Qty): Envía un requerimiento al servidor para “Read Coils” acorde a la función número 1 del protocolo ModBus.		
Argumentos	**Tipo**	**Descripción**
StartAddr	E	Dirección de la primera salida a leer.
Qty	E	Cantidad de salidas a leer (máximo 256).
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa, requerimiento enviado.
-1	S	Error, librería ocupada.
-2	S	Error, no se puede asignar conexión.
-4	S	Error, la librería no fue inicializada.
-20	S	Error, el valor del argumento **Qty** es invalido.
Notas		**Descripción**
1		Esta función establece una conexión con el servidor ModBus si no la hay, y luego envía el requerimiento.
2		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbClGetLibStatus() o asincrónicamente desde el evento @OnMbClientRx().
3		Con la función MbClGetRxReg() es posible leer los datos recibidos y con la función MbClGetExceptionCode() puede obtener la excepción ModBus retornada.

Ejemplo 1:

```
// Enviar requerimiento al servidor ModBus para leer
// 8 salidas discretas a partir de la dirección 1.

if( MbClSendReadCoils(1, 8) < 0 )
{
   // Error.
}
```

MbClSendReadDiscretes(StartAddr, Qty): Envía un requerimiento al servidor para “Read Discrete Inputs” acorde a la función número 2 del protocolo ModBus.		
Argumentos	**Tipo**	**Descripción**
StartAddr	E	Dirección de la primera entrada a leer.
Qty	E	Cantidad de entradas a leer (máximo 256).
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa, requerimiento enviado.
-1	S	Error, librería ocupada.
-2	S	Error, no se puede asignar conexión.
-4	S	Error, la librería no fue inicializada.
-20	S	Error, el valor del argumento **Qty** es invalido.
Notas		**Descripción**
1		Esta función establece una conexión con el servidor ModBus si no la hay, y luego envía el requerimiento.
2		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbClGetLibStatus() o asincrónicamente desde el evento @OnMbClientRx().
3		Con la función MbClGetRxReg() es posible leer los datos recibidos y con la función MbClGetExceptionCode() puede obtener la excepción ModBus retornada.

Ejemplo 1:

```
// Enviar requerimiento al servidor ModBus para leer
// 4 entradas discretas a partir de la dirección 10001.

if( MbClSendReadDiscretes(10001, 4) < 0 )
{
   // Error.
}
```

MbClSendReadHoldingReg (StartAddr, Qty): Envía un requerimiento al servidor para “Read Holding Registers” acorde a la función número 3 del protocolo ModBus.		
Argumentos	**Tipo**	**Descripción**
StartAddr	E	Dirección del primero registro de 16-bits a leer.
Qty	E	Cantidad de registros a leer (máximo 16).
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa, requerimiento enviado.
-1	S	Error, librería ocupada.
-2	S	Error, no se puede asignar conexión.
-4	S	Error, la librería no fue inicializada.
-20	S	Error, el valor del argumento **Qty** es invalido.
Notas		**Descripción**
1		Esta función establece una conexión con el servidor ModBus si no la hay, y luego envía el requerimiento.
2		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbClGetLibStatus() o asincrónicamente desde el evento @OnMbClientRx().
3		Con la función MbClGetRxReg() es posible leer los datos recibidos y con la función MbClGetExceptionCode() puede obtener la excepción ModBus retornada.

Ejemplo 1:

```
// Enviar requerimiento al servidor ModBus para leer
// 6 registros a partir de la dirección 40001.

if( MbClSendReadHoldingReg(40001, 6) < 0 )
{
   // Error.
}
```

MbClSendReadInputReg (StartAddr, Qty): Envía un requerimiento al servidor para “Read Input Registers” acorde a la función número 4 del protocolo ModBus.		
Argumentos	**Tipo**	**Descripción**
StartAddr	E	Dirección del primero registro de 16-bits a leer.
Qty	E	Cantidad de registros a leer (máximo 16).
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa, requerimiento enviado.
-1	S	Error, librería ocupada.
-2	S	Error, no se puede asignar conexión.
-4	S	Error, la librería no fue inicializada.
-20	S	Error, el valor del argumento **Qty** es invalido.
Notas		**Descripción**
1		Esta función establece una conexión con el servidor ModBus si no la hay, y luego envía el requerimiento.
2		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbClGetLibStatus() o asincrónicamente desde el evento @OnMbClientRx().
3		Con la función MbClGetRxReg() es posible leer los datos recibidos y con la función MbClGetExceptionCode() puede obtener la excepción ModBus retornada.

Ejemplo 1:

```
// Enviar requerimiento al servidor ModBus para leer
// 6 registros a partir de la dirección 40001.

if( MbClSendReadInputReg(30001, 6) < 0 )
{
   // Error.
}
```

MbClSendWriteCoil(StartAddr, Value): Envía un requerimiento al servidor para “Write Single Coil” acorde a la función número 5 del protocolo ModBus.		
Argumentos	**Tipo**	**Descripción**
Addr	E	Dirección de la salida a escribir.
Value	E	Valor a escribir. El valor 0 desactiva la salida y el valor 0xFF00 activa la salida.
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa, requerimiento enviado.
-1	S	Error, librería ocupada.
-2	S	Error, no se puede asignar conexión.
-4	S	Error, la librería no fue inicializada.
Notas		**Descripción**
1		Esta función establece una conexión con el servidor ModBus si no la hay, y luego envía el requerimiento.
2		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbClGetLibStatus() o asincrónicamente desde el evento @OnMbClientRx().
3		Con la función MbClGetRxReg() es posible leer los datos recibidos y con la función MbClGetExceptionCode() puede obtener la excepción ModBus retornada.

Ejemplo 1:

```
// Enviar requerimiento al servidor ModBus para activar
// la salida 6.

if( MbClSendWriteCoil(6, 0xFF00) < 0 )
{
   // Error.
}
```

MbClSendWriteReg(StartAddr, Value): Envía un requerimiento al servidor para "Write Single Register" acorde a la función número 6 del protocolo ModBus.		
Argumentos	**Tipo**	**Descripción**
Addr	E	Dirección del registro a escribir.
Value	E	Valor a escribir. Valor de 16-bits, entre 0 y 65535.
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa, requerimiento enviado.
-1	S	Error, librería ocupada.
-2	S	Error, no se puede asignar conexión.
-4	S	Error, la librería no fue inicializada.
Notas		**Descripción**
1		Esta función establece una conexión con el servidor ModBus si no la hay, y luego envía el requerimiento.
2		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbClGetLibStatus() o asincrónicamente desde el evento @OnMbClientRx().
3		Con la función MbClGetRxReg() es posible leer los datos recibidos y con la función MbClGetExceptionCode() puede obtener la excepción ModBus retornada.

Ejemplo 1:

```
// Enviar requerimiento al servidor ModBus y escribir
// el valor 666 en el registro 40001.

if( MbClSendWriteReg(40001, 666) < 0 )
{
   // Error.
}
```

MbClSendWriteMultCoils(StartAddr, Qty, Values[]): Envía un requerimiento al servidor para "Write Multiple Coils" acorde a la función número 15 del protocolo ModBus.		
Argumentos	**Tipo**	**Descripción**
StartAddr	E	Dirección de primera salida a escribir.
Qty	E	Cantidad de salidas a escribir. Máximo 256.
Values[]	E	Array con los valores a escribir. De cada elemento del array, solo se toman los 8 bits menos significativos, es decir solo 1 byte por elemento. El array debe tener Qty / 8 elementos (sumar 1 si la división da menor a 1).
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa, requerimiento enviado.
-1	S	Error, librería ocupada.
-2	S	Error, no se puede asignar conexión.
-4	S	Error, la librería no fue inicializada.
-10	S	Error, dirección del array inválida.
-20	S	Error, valor **Qty** inválido.
Notas		**Descripción**
1		Esta función establece una conexión con el servidor ModBus si no la hay, y luego envía el requerimiento.
2		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbClGetLibStatus() o asincrónicamente desde el evento @OnMbClientRx().
3		Con la función MbClGetRxReg() es posible leer los datos recibidos y con la función MbClGetExceptionCode() puede obtener la excepción ModBus retornada.

Ejemplo 1:

```
// Inicializar un array con los siguientes valores.
new Values[] = {0xFF, 0x0F}

// Enviar requerimiento al servidor ModBus y escribir
// 16 salidas a partir de la dirección 1. Las salidas
// 1 a 12 se activaran con el valor 1, y las salidas 13 a 16
// se desactivaran con el valor 0.

if(MbClSendWriteMultCoils (1, 16, Values) < 0 )
{
   // Error.
}
```

MbClSendWriteMultReg(StartAddr, Qty, Values[]): Envía un requerimiento al servidor para "Write Multiple Registers" acorde a la función número 16 del protocolo ModBus.		
Argumentos	**Tipo**	**Descripción**
StartAddr	E	Dirección del primer registro a escribir.
Qty	E	Cantidad de registros a escribir. Máximo 16.
Values[]	E	Array con los valores a escribir. De cada elemento del array, solo se toman los 16 bits menos significativos, es decir solo 2 bytes por elemento. El array debe tener Qty elementos.
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa, requerimiento enviado.
-1	S	Error, librería ocupada.
-2	S	Error, no se puede asignar conexión.
-4	S	Error, la librería no fue inicializada.
-10	S	Error, dirección del array inválida.
-20	S	Error, valor **Qty** inválido.
Notas		**Descripción**
1		Esta función establece una conexión con el servidor ModBus si no la hay, y luego envía el requerimiento.
2		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbClGetLibStatus() o asincrónicamente desde el evento @OnMbClientRx().
3		Con la función MbClGetRxReg() es posible leer los datos recibidos y con la función MbClGetExceptionCode() puede obtener la excepción ModBus retornada.

Ejemplo 1:

```
// Inicializar un array con los siguientes valores.
new Values[] = {111, 222, 333, 444}

// Enviar requerimiento al servidor ModBus y escribir
// 4 registros a partir de la dirección 40001. En el
// servidor el registro 40001 tendra el valor 111,
// el registro 40002 el valor 222, etc.

if(MbClSendWriteMultReg(40001, 4, Values) < 0 )
{
   // Error.
}
```

MbClSendReadWriteMultReg(RdStartAddr, RdQty, WrStartAddr, WrQty, Values[ ]): Envía un requerimiento al servidor para "Write Read Multiple Registers" acorde a la función número 23 del protocolo ModBus.		
Argumentos	**Tipo**	**Descripción**
RdStartAddr	E	Dirección del primer registro a leer.
RdQty	E	Cantidad de registros a leer. Máximo 16.
WrStartAddr	E	Dirección del primer registro a escribir.
WrQty	E	Cantidad de registros a escribir. Máximo 16.
Values[]	E	Array con los valores a escribir. De cada elemento del array, solo se toman los 16 bits menos significativos, es decir solo 2 bytes por elemento. El array debe tener WrQty elementos.
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa, requerimiento enviado.
-1	S	Error, librería ocupada.
-2	S	Error, no se puede asignar conexión.
-4	S	Error, la librería no fue inicializada.
-10	S	Error, dirección del array inválida.
-20	S	Error, valor **RdQty o WrQty** inválido.
Notas		**Descripción**
1		Esta función establece una conexión con el servidor ModBus si no la hay, y luego envía el requerimiento.
2		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbClGetLibStatus() o asincrónicamente desde el evento @OnMbClientRx().
3		Con la función MbClGetRxReg() es posible leer los datos recibidos y con la función MbClGetExceptionCode() puede obtener la excepción ModBus retornada.

Ejemplo 1:

```
// Inicializar un array con los siguientes valores.
new Values[] = {111, 222, 333, 444}

// Enviar requerimiento al servidor ModBus y leer
// 5 registros a partir de la dirección 40001. Al
// mismo tiempo escribir 4 registros a partir de la
// dirección 40001 con los valores del array Values[].

if( MbClSendReadWriteMultReg(40001, 5, 40001, 4, Values) < 0 )
{
   // Error.
}
```

MbClClose(): Cierra una conexión establecida con el servidor ModBus. Si la conexión no fue establecida, la función no tiene efecto y devuelve un código de error.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
0	S	Conexión cerrada con éxito.
-1	S	Error, no hay conexión establecida con el servidor.
-2	S	Error, no hay conexión asignada, número máximo de conexiones alcanzado.
-4	S	Error, la librería no fue inicializada.
Notas		**Descripción**
1		Esta función se utiliza para cerrar una conexión con un servidor ModBus, informándole explícitamente que el cliente se desconecto. Se recomienda utilizar esta función al cambiar de servidor o cuando no utilizamos mas la conexión.
2		Si se envió un requerimiento al servidor ModBus, pero este no existe porque está apagado (por ejemplo), utilizar esta función no cierra la conexión, ya que la misma no existe y devuelve error. El cliente esperará un tiempo (55 segundos aproximadamente) y luego informara que existe un timeout.

Ejemplo 1:

```
// Cerrar formalmente la conexión actual.

if(MbClClose() < 0 )
{
   // Error, verifique código de error.
}
```

6.1.3 Funciones para leer Respuestas, Estado y Errores

Las siguientes funciones le permiten determinar el estado de la transacción, leer respuestas del servidor ModBus o determinar errores.

MbClGetLibStatus(): Obtiene un código de estado actual de la librería, es decir del cliente ModBus. Siempre debe comprobar el código de estado antes de realizar una operación. Un código de estado negativo, implica una condición de error. Un código positivo implica un estado particular. El valor cero significa que el cliente recibió respuesta del servidor. Ver Tabla 4 en página 42 para constantes.

Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
127	S	Requerimiento en proceso de envío.
126	S	La librería no fue inicializada aun.
125	S	Respuesta recibida, pero no procesada.
124	S	La librería fue inicializada.
123	S	La conexión fue cerrada por el cliente.
0	S	Respuesta recibida con éxito del servidor, listo.
-1	S	Error, conexión abortada.
-2	S	Error, timeout en conexión. Usualmente cuando el servidor no existe.
-3	S	Error, conexión cerrada por el host.
-4	S	Error, timeout en transacción. Especificado por función de inicialización.
-5	S	Error, la conexión no fue asignada.
-6	S	Error, ver código de retorno de función.
-7	S	Error, se recibió un código ModBus de excepción.
-8	S	Error, TID (transaction ID) recibido difiere.
-9	S	Error, UID (unit ID) recibido difiere.
-10	S	Error, código de función ModBus recibido difiere al enviado.
-11	S	Error, PID (protocol ID) recibido incorrecto.
Notas		**Descripción**
1		Los códigos más importantes son el 127 (cuando hay una transacción en curso) y el 0 (cuando la respuesta del servidor está disponible). Para comprobar una situación de error, solo hay que verificar si el código devuelto es negativo.

Ejemplo 1:

```
// Crear variable que indica si esperamos una respuesta del Servidor.
new WaitResponse = 0

// Variable para almacenar datos.
new RxData[4]

// Crear variable para almacenar estado.
new MbStat

// Inicializar Cliente ModBus TCP.
MbClInit(IpToVar(192,168,1,15), 502, 30, 0)

for(;;)
{
   // Realizar transaccion ModBus si DIN5 = 1.
   if(DinValue(DIN5) && WaitResponse == 0)
   {
      // Enviar peticion para leer "Holding Registers".
      if(MbClSendReadHoldingReg(0, 4) < 0)
      {
         // Error en transmision.
      }
      else
      {
         WaitResponse = 1
      }
   }

   // Obtener Estado de Libreria.
   MbStat = MbClGetLibStatus()

   // Comprobar si llego respuesta del Server.
   if(MbStat == 0 && WaitResponse == 1)
   {
        // Si, leer registros recibidos (byte-offset = 0).
        MbClGetRxReg(RxData, 0, 4, 0)

        // Mostrar valores de registros en LCD (opcional).

        // No se espera respuesta.
        WaitResponse = 0
   }

   // Comprobar errores.
   if(MbStat < 0)
   {
        // Error!... ver código de error en MbStat.
        WaitResponse = 0
   }
}
```

El código de la página anterior, es un programa completo para leer registros holding en un servidor ModBus que se encuentra en la dirección IP 192.168.1.15.

El código empieza definiendo variables e inicializando el cliente ModBus con los datos del servidor remoto.

Luego en el loop infinito principal, se comprueba si la entrada **DIN5** del PLC está en nivel "1" y si el cliente ModBus no está esperando ninguna respuesta, es decir si **WaitResponse** vale 0.

Si **DIN5**=1 y **WaitResponse**=0, se procede a enviar un requerimiento al servidor con **MbClSendReadHoldingReg()**.

Luego se lee el código de estado de la librería con **MbClGetLibStatus()** y se almacena en la variable **MbStat**. A continuación se comprueba si el código es 0 (respuesta recibida), en caso afirmativo se emplea **MbClGetRxReg()** para obtener los valores de los registros recibidos y copiarlos en la variable **RxData[ ]**.

Si el código de estado es negativo, el código lo comprueba y puede emitir una indicación de error.

Utilizamos la variable "**WaitResponse**" como flag para indicar en el programa si estamos esperando una respuesta, de esta forma evitamos llamar a **MbClSendReadHoldingReg()** mientras la transacción está en curso (de otra forma la función retornaría -1, ya que la librería está ocupada con la transacción pendiente).

La siguiente tabla muestra nombres de constantes que pueden utilizarse para comprobar valores retornados por **MbClGetLibStatus()**, y así evitar el uso de valores numéricos.

Tabla 4: Códigos de estados retornado por MbClGetLibStatus()

Nombre	Valor	Descripción
MBCL_STAT_TRY_TO_SEND	127	Requerimiento en proceso de envío.
MBCL_STAT_LIB_NOT_INIT	126	La librería no fue inicializada aun.
MBCL_STAT_RESP_RXED	125	Respuesta recibida, pero no procesada.
MBCL_STAT_INITIALIZED	124	La librería fue inicializada.
MBCL_STAT_CLOSED	123	La conexión fue cerrada por el cliente.
MBCL_STAT_OK	0	Respuesta recibida con éxito del servidor, listo.
MBCL_STAT_ERR_ABORTED	-1	Error, conexión abortada.
MBCL_STAT_ERR_CONN_TO	-2	Error, timeout en conexión. Usualmente cuando el servidor no existe.
MBCL_STAT_ERR_CLOSED	-3	Error, conexión cerrada por el host.
MBCL_STAT_ERR_TO	-4	Error, timeout en transacción. Especificado por función de inicialización.
MBCL_STAT_ERR_ALLOC	-5	Error, la conexión no fue asignada.
MBCL_STAT_ERR_FUNC	-6	Error, ver código de retorno de función.
MBCL_STAT_ERR_EXCEP	-7	Error, se recibió un código ModBus de excepción.
MBCL_STAT_ERR_BAD_TID	-8	Error, TID (transaction ID) recibido difiere.
MBCL_STAT_ERR_BAD_UID	-9	Error, UID (unit ID) recibido difiere.
MBCL_STAT_ERR_FUNC_DIF	-10	Error, código de función ModBus recibido difiere al enviado.
MBCL_STAT_ERR_BAD_PID	-11	Error, PID (protocol ID) recibido incorrecto.

Ejemplo:

```
// Comprobar si llego respuesta del Server.
// Notar como reemplazamos el numero 0 por la constante equivalente.
if(MbClGetLibStatus() == MBCL_STAT_OK && WaitResponse == 1)
{
     // Si, leer registros recibidos.
     MbClGetRxReg(RxData, 0, 4, 0)

     // Mostrar valores de registros en LCD (opcional).

     // No se espera respuesta.
     WaitResponse = 0
}
```

MbClGetRxReg(Data[ ], Index, Max, ByteOffset): Obtiene los datos en forma de registros de la ultima respuesta recibida del servidor ModBus.		
Argumentos	**Tipo**	**Descripción**
Data[]	S	Array donde se copiaran los datos recibidos. En cada elemento del array se guardan 2 bytes de la respuesta recibida interpretados como un registros de 16-bits.
Index	E	Índice dentro del array donde comienza la copia.
Max	E	Número máximo de bytes / registros a copiar. Máximo valor (32 o 16).
ByteOffset	E	Offset en bytes a partir donde se comienzan a leer los registros de la respuesta ModBus. Muy útil cuando al comienzo de la trama hay información de la respuesta, pero no son propiamente los registros de 16-bits devueltos.
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa, datos copiados.
-1	S	Error, no hay datos disponibles para copiar.
-2	S	Error, número incorrecto de datos a copiar.
-4	S	Error, dirección del array inválida.
-5	S	Error, valor invalido de ByeOffset.
Notas		**Descripción**
1		Antes de leer en buffer, es útil verificar el estado de la librería, ver Tabla 4 en página 42 para constantes.

Ejemplo 1:

```
new RxData[4]

// Comprobar si llego respuesta del Server.
if(MbClGetLibStatus() == MBCL_STAT_OK)
{
     // Si, leer registros recibidos, byte-offset = 1.
     MbClGetRxReg(RxData, 0, 4, 1)
}
```

Ejemplo 2:

```
new RxData[4]

// Comprobar si llego respuesta del Server.
if(MbClGetLibStatus() == MBCL_STAT_OK)
{
     // Si, leer registros recibidos, sin byte-offset.
     MbClGetRxReg(RxData, 0, 4, 0)
}
```

MbClGetRxBytes(Data[ ], Index, Max, ByteOffset): Obtiene los datos en forma de bytes de la ultima respuesta recibida del servidor ModBus.		
Argumentos	**Tipo**	**Descripción**
Data[]	S	Array donde se copiaran los datos recibidos. En cada elemento del array se guarda 1 byte de la respuesta recibida.
Index	E	Índice dentro del array donde comienza la copia.
Max	E	Número máximo de bytes / registros a copiar. Máximo valor (32 o 16).
ByteOffset	E	Offset en bytes a partir donde se comienzan a leer los bytes de la respuesta ModBus.
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa, datos copiados.
-1	S	Error, no hay datos disponibles para copiar.
-2	S	Error, número incorrecto de datos a copiar.
-4	S	Error, dirección del array inválida.
-5	S	Error, valor inválido de ByeOffset.
Notas		**Descripción**
1		Antes de leer en buffer, es útil verificar el estado de la librería, ver Tabla 4 en página 42 para constantes.

Ejemplo:

```
new RxData[4]

// Comprobar si llego respuesta del Server.
if(MbClGetLibStatus() == MBCL_STAT_OK)
{
     // Si, leer 4 bytes recibidos, sin byte-offset.
     MbClGetRxBytes(RxData, 0, 4, 0)
}
```

MbClGetFuncCode(): Obtiene la función ModBus a la que pertenece la ultima respuesta recibida del servidor ModBus.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
>= 0	S	Ultima función ModBus recibida, valor igual o mayor a 0.
-4	S	Error, librería no inicializada.
Notas		**Descripción**
-		

Ejemplo:

```
new FuncCode

// Obtener código de función de ultima respuesta.
FuncCode = MbClGetFuncCode()
```

El código de función retornado corresponde al valor ModBus definido por el estándar. El entorno StxLadder ya incluye las siguientes constantes, que puede utilizar para fines de comparación.

Tabla 5: Constantes para código de funciones ModBus (se listan solo las mas comunes)

Nombre	Valor
MB_FUNC_NONE	0
MB_FUNC_READ_COILS	1
MB_FUNC_READ_DISCRETE_INPUTS	2
MB_FUNC_WRITE_SINGLE_COIL	5
MB_FUNC_WRITE_MULTIPLE_COILS	15
MB_FUNC_READ_HOLDING_REGISTER	3
MB_FUNC_READ_INPUT_REGISTER	4
MB_FUNC_WRITE_REGISTER	6
MB_FUNC_WRITE_MULTIPLE_REGISTERS	16
MB_FUNC_READWRITE_MULTIPLE_REGISTERS	23

MbClGetExceptionCode(): Obtiene la excepción ModBus de la ultima respuesta recibida del servidor ModBus.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
>= 0	S	Ultima excepción ModBus recibida, valor igual o mayor a 0.
-4	S	Error, librería no inicializada.
Notas		**Descripción**
-		

Ejemplo:

```
new ExpCode

// Obtener código de excepción de ultima respuesta.
ExpCode = MbClGetExceptionCode()
```

El código de excepción retornado corresponde al valor ModBus definido por el estándar. El entorno StxLadder ya incluye las siguientes constantes, que puede utilizar para fines de comparación.

Tabla 6: Constantes para código de excepciones ModBus (se listan solo las más comunes)

Nombre	Valor Hex
MB_EX_NONE	0
MB_EX_ILLEGAL_FUNCTION	1
MB_EX_ILLEGAL_DATA_ADDRESS	2
MB_EX_ILLEGAL_DATA_VALUE	3
MB_EX_SLAVE_DEVICE_FAILURE	4
MB_EX_ACKNOWLEDGE	5
MB_EX_SLAVE_BUSY	6
MB_EX_MEMORY_PARITY_ERROR	8
MB_EX_GATEWAY_PATH_FAILED	A
MB_EX_GATEWAY_TGT_FAILED	B

Es recomendado ver la **Tabla 2: Excepciones ModBus (Resumen)** en pagina 4.

MbClGetDataLength(): Obtiene la cantidad de bytes recibidos de la ultima respuesta del servidor ModBus.		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
>= 0	S	Cantidad de bytes, valor igual o mayor a 0.
-4	S	Error, librería no inicializada.
Notas		**Descripción**
1		Corresponde al campo LENGTH del MBAP en paquete ADU (Aplication Data Unit).
2		Solo para usos de depuración.

Ejemplo:

```
new Length

// Obtener cantidad de bytes de ultima respuesta.
Length = MbClGetDataLength()
```

6.1.4 Funciones para Eventos

Las siguientes funciones le permiten activar o desactivar eventos.

MbClSetRxEvent(): Activa el evento @OnMbClientRx().		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, el evento no pudo ser creado.
Notas		**Descripción**
1		El evento @OnMbClientRx() se activa cuando una transacción termina y el código de estado de la librería (ver Tabla 4 en página 42 para constantes) es igual o menor a 0.
2		El uso de eventos, le permite utilizar de forma asíncrona las funciones para realizar requerimientos al servidor ModBus.

Ejemplo:

El siguiente ejemplo envía una petición para leer “Holding Registers” al servidor ModBus cuando la entrada DIN5 tiene el valor “1”. La respuesta del servidor se lee cuando se genera el evento @OnMbClientRx(), de esta forma el código principal puede realizar otras tareas.

```
// -------------------------------------------------------------------------
// Funcion: PlcMain ()
// Funcion principal.
// -------------------------------------------------------------------------

PlcMain()
{
   // Inicializar display.
   LcdClear()
   LcdPrintf(0,1, "ModBus TCP")

   // Inicializar Cliente ModBus TCP.
   MbClInit(IpToVar(192,168,1,15), 502, 0)

   // Activar evento "OnMbClientRx" para determinar respuesta
   // del servidor.
   if(MbClSetRxEvent() < 0)
   {
            LcdClear()
            LcdPrintf(0,1, "Error, Evento")
            DelayS(5)
   }
```

```
for(;;)
   {
      // Realizar transaccion ModBus si DIN5 = 1.
      if(DinValue(DIN5) && WaitResponse == 0)
      {
               WaitResponse = 1

               LcdClear()
               LcdPrintf(0,1, "Enviando ...")

               // Enviar peticion para leer "Holding Registers".
               if(MbClSendReadHoldingReg(40001, 4) < 0)
               {
                     LcdClear()
                     LcdPrintf(0,1, "Send error...")

                     WaitResponse = 0
               }
       }
   }

   // Retorno.
   return 0
}

// ---------------------------------------------------------------------------
// Funcion: @OnMbClientRx()
// Procesa una respuesta del servidor ModBus TCP (escritura/lectura).
// ---------------------------------------------------------------------------

@OnMbClientRx()
{
      new MbStat

      // Obtener Estado de Libreria.
      MbStat = MbClGetLibStatus()

      // Mostrar Estado de Libreria.
      LcdPrintf(0,0, "Stat = %03d", MbStat)

      // Leer registros recibidos si no hay error.
      if(MbStat == 0)
      {
         // Leer Registros (byte-offset=0).
         MbClGetRxReg(RxData, 0, 4, 0)

         // Mostrar valores de registros en LCD.
         LcdPrintf(0,1, "%04d %04d %04d", RxData[0], RxData[1], RxData[2])

         // Respuesta recibida, no esperar mas.
         WaitResponse = 0
      }
}
```

MbClClrRxEvent(): Desactiva el evento @OnMbClientRx().		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, el evento no pudo ser desactivado.
Notas		**Descripción**
-		

Ejemplo:

```
// Desactivar evento "@OnMbClientRx”.
if(MbClClrRxEvent() < 0)
{
   // Error.
}
```

7 Abreviaciones y Términos Empleados

- **PLC**: Programable Logic Controller (Controlador Lógico Programable).
- **IP**: Dirección Internet, conformada por cuatro octetos, por ejemplo 192.168.1.81.
- **MB:** ModBus.
- **Ethernet**: Red de computadoras, que generalmente se utilizan el protocolo de internet TCP/IP o UDP/IP.
- **Transacción**: Proceso de enviar un requerimiento y esperar respuesta de un servidor ModBus.

8 Historial de Revisiones

Tabla 7: Historia de Revisiones del Documento

Revisión	Cambios	Descripción	Estado
03 13/May/2015	4	1. Se agrega soporte y documentación para Lenguaje Ladder. 2. Se corrigen errores de tipeo en funciones Pawn y descripciones. 3. Agrega funciones MbClGetRxReg() y MbClGetRxBytes(). 4. Se elimina descripción de función MbClGetRxData().	Preliminar
02 06/APR/2013	1	1. Se cambia dirección 45XXX por 40XXX en ejemplos.	Preliminar
01 01/MAR/2013	1	1. Versión preliminar liberada.	Preliminar

9 Referencias

Ninguna.

10 Información Legal

10.1 Aviso de exención de responsabilidad

General: La información de este documento se da en buena fe, y se considera precisa y confiable. Sin embargo, Slicetex Electronics no da ninguna representación ni garantía, expresa o implícita, en cuanto a la exactitud o integridad de dicha información y no tendrá ninguna responsabilidad por las consecuencias del uso de la información proporcionada.

El derecho a realizar cambios: Slicetex Electronics se reserva el derecho de hacer cambios en la información publicada en este documento, incluyendo, especificaciones y descripciones de los productos, en cualquier momento y sin previo aviso. Este documento anula y sustituye toda la información proporcionada con anterioridad a la publicación de este documento.

Idoneidad para el uso: Los productos de Slicetex Electronics no están diseñados, autorizados o garantizados para su uso en aeronaves, área médica, entorno militar, entorno espacial o equipo de apoyo de vida, ni en las aplicaciones donde el fallo o mal funcionamiento de un producto de Slicetex Electronics pueda resultar en lesiones personales, muerte o daños materiales o ambientales graves. Slicetex Electronics no acepta ninguna responsabilidad por la inclusión y / o el uso de productos de Slicetex Electronics en tales equipos o aplicaciones (mencionados con anterioridad) y por lo tanto dicha inclusión y / o uso es exclusiva responsabilidad del cliente.

Aplicaciones: Las aplicaciones que aquí se describen o por cualquiera de estos productos son para fines ilustrativos. Slicetex Electronics no ofrece representación o garantía de que dichas aplicaciones serán adecuadas para el uso especificado, sin haber realizado más pruebas o modificaciones.

Los valores límites o máximos: Estrés por encima de uno o más valores límites (como se define en los valores absolutos máximos de la norma IEC 60134) puede causar daño permanente al dispositivo. Los valores límite son calificaciones de estrés solamente y el funcionamiento del dispositivo en esta o cualquier otra condición por encima de las indicadas en las secciones de Características de este documento, no está previsto ni garantizado. La exposición a los valores limitantes por períodos prolongados puede afectar la fiabilidad del dispositivo.

Documento: Prohibida la modificación de este documento en cualquier medio electrónico o impreso, sin autorización previa de Slicetex Electronics por escrito.

11 Información de Contacto

Para mayor información, visítenos en www.slicetex.com

Para ventas e información general, envíe un mail a: info@slicetex.com

Para soporte técnico ingrese en: www.slicetex.com/foro

Slicetex Electronics
Córdoba, Argentina

12 Contenido

12.1 Índice de Tablas