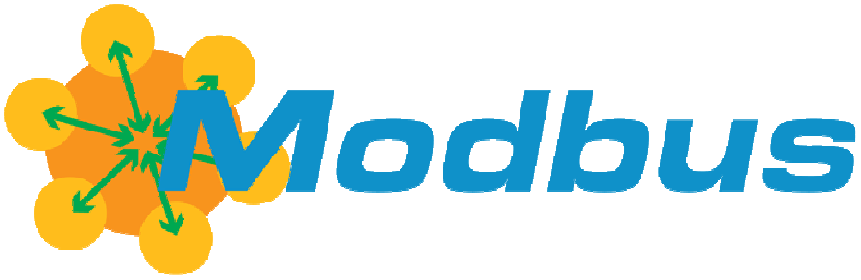
Slicetex Ladder Designer Studio

NOTA DE APLICACIÓN

AN026
ModBus RTU – 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 RTU** 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.*

***ModBusRTU** permite el empleo del protocolo original **ModBus** con dispositivos que se comunican mediante una interfaz serial, por ejemplo RS-232, RS-485 o simplemente lógica TTL.*

*Este documento detalla el uso del protocolo **ModBus RTU** en modo cliente (Master), que le permitirá conectar el PLC a diferentes dispositivos **ModBus RTU** 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 RTU:

http://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf

2.1 Ejemplos

En nuestro sitio web, busque la pagina de la nota de aplicación AN026, 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 RTU** como cliente. La comunicación se realiza a través de una interfaz serie tipo RS-232, RS-485 o simplemente TTL (esto depende de las capacidades de hardware del PLC y/o los módulos de expansión conectados).

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

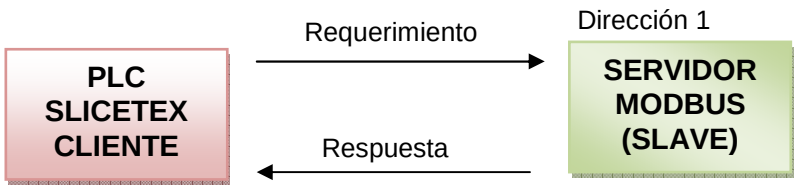


Fig. 1: Transacción ModBus RTU.

La figura 1, muestra el proceso que lleva a cabo el PLC cuando se realiza una transacción ModBus RTU 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, por ejemplo la dirección ModBus RTU, velocidad del puerto, formato de datos, etc.
2. **Requerimiento**: El PLC envía una función ModBus al servidor para leer o escribir datos.
3. **Respuesta**: El servidor procesa el requerimiento del cliente, y devuelve una respuesta que depende de la función ModBus ejecutada.
4. **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.
5. **Fin:** El cliente puede realizar otro requerimiento al servidor si es necesario.

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 inválidos.
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.

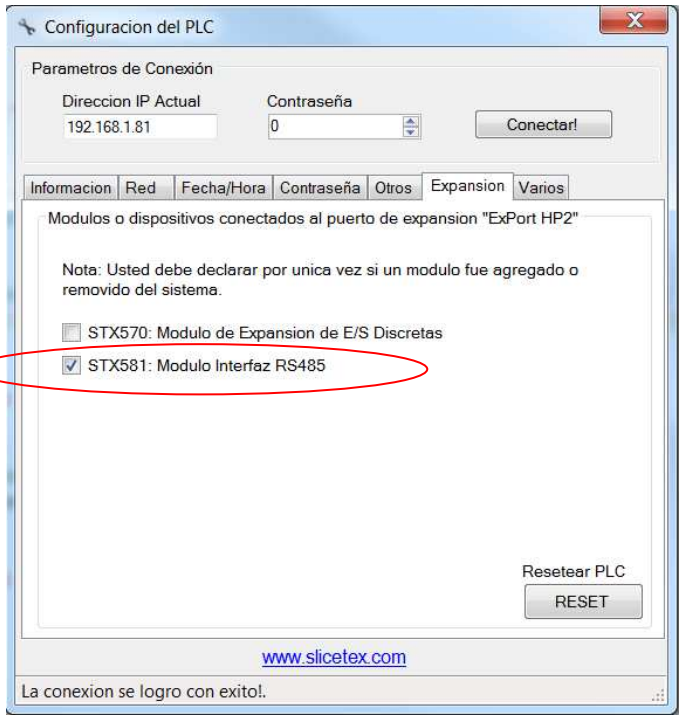
5 Selección de Interface Física en PLC

Con algunos modelos de PLC o módulos de expansión, es necesario declarar que interface física utilizará el protocolo ModBus RTU para las transacciones, de tal forma que el PLC pueda inicializar correctamente el hardware asociado. Consulte hoja de datos del dispositivo.

Por ejemplo, si se utiliza el modulo de expansión **STX581**, que añade una interface RS-485 cuando se conecta al módulo de expansión del PLC, es necesario desde el entorno **StxLadder**, configurar el PLC como se muestra a continuación:

Ir a menú "**PLC -> Configurar PLC**". Presionar el botón "**Conectar!**" para conectarse al PLC.

Una vez conectado, dirigirse a la pestaña "Expansión" y seleccionar el check-box "**STX581: Modulo Interfaz RS485**", como se muestra a continuación:



Resetear una vez configurado el PLC.

Esto permite utilizar la interfaz RS-485 con un modulo STX581 conectado al puerto de expansión.

Si usted está utilizando un modulo de expansión RS-232 (como la placa STX580) o simplemente señales TTL del puerto de expansión del PLC, no hace falta declarar nada aquí.

Igualmente, algunos modelos de PLC ya pueden tener integrado el puerto RS-485. Consulte en nuestro foro para más información.

6 ModBus Cliente con Lenguaje Ladder

Los componentes ModBus para Lenguaje Ladder están en elaboración, pronto los publicaremos.

7 ModBus Cliente con Lenguaje Pawn

En esta sección explicaremos a modo general como utilizar el protocolo ModBus RTU 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.

En lenguaje Pawn es muy simple conectarse al servidor ModBus RTU. 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.

7.1 Funciones Nativas en Pawn Disponibles

7.1.1 Funciones de Configuración

MbRtuClInit(Baudrate, FormatMode, Timeout, Interface): Inicializa y configura los parámetros para que el PLC pueda conectarse a un servidor ModBus RTU cuando realice una transacción. Esta función debe llamarse una sola vez, antes de cualquier otra función ModBus RTU.		
Argumentos	**Tipo**	**Descripción**
Baudrate	E	Velocidad en bit-por-segundos de la conexión. Debe coincidir con la velocidad del servidor. En general 9600 (preferentemente) o 19200. Valor máximo 115200.
FormatMode	E	Formato de los datos serie. Se utiliza por lo general "**SERIAL_8E1**", es decir 8-bits de datos, paridad par (even) y 1-bit de stop. Otros valores comunes pueden ser "**SERIAL_8N1**" o "**SERIAL_8N2**".
Timeout	E	Tiempo en segundos que debe esperar el cliente espere respuesta a un requerimiento. Mínimo 1 y Máximo 255.
Interface	E	Interface utilizada para la comunicación ModBus. Utilice las constantes: • MB_RTU_INTERFACE_RS232: RS-232. • MB_RTU_INTERFACE_RS485: RS-485.
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, el cliente ya fue inicializado.
-2	S	Error, falla en inicialización, memoria insuficiente.
-4	S	Error, en inicialización de hardware.
-5	S	Error, Baudrate inválido.
Notas		**Descripción**
1		El valor de timeout es ignorado si se realiza una transacción a dirección Broadcast.
2		Algunos modelos de PLC o módulos de expansión conectados pueden requerir configuración extra para la interfaz eléctrica seleccionada. Consulte hoja de datos del dispositivo.

Ejemplo 1:

```
new FuncStat

// Inicializar Cliente ModBus RTU.
// Velocidad 9600 bps, formato 8E1, timeout 10 segundos, interface RS232.

FuncStat = MbRtuClInit(9600, SERIAL_8E1, 10, MB_RTU_INTERFACE_RS232)

if(FuncStat < 0)
{
      LcdClear()
      LcdPrintf(0,1, "Init err: %d", FuncStat)
}
```

7.1.2 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.

MbRtuClSendReadCoils(SlaveAddr, StartAddr, Qty): Envía un requerimiento al servidor para "Read Coils" acorde a la función número 1 del protocolo ModBus.		
Argumentos	**Tipo**	**Descripción**
SlaveAddr	E	Dirección ModBus RTU del servidor.
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.
-4	S	Error, la librería no fue inicializada.
-20	S	Error, el valor del argumento **Qty** es invalido.
Notas		**Descripción**
1		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbRtuClGetLibStatus() o asincrónicamente desde el evento @OnMbRtuClientRx().
2		Con la función MbRtuClGetRxReg() es posible leer los datos recibidos y con la función MbRtuClGetExceptionCode() puede obtener la excepción ModBus retornada.

Ejemplo 1:

```
// Enviar requerimiento al servidor ModBus RTU (dirección 5) para leer
// 8 salidas discretas a partir de la dirección 1 de memoria.

if( MbRtuClSendReadCoils(5, 1, 8) < 0 )
{
   // Error.
}
```

MbRtuClSendReadDiscretes(SlaveAddr, StartAddr, Qty): Envía un requerimiento al servidor para "Read Discrete Inputs" acorde a la función número 2 del protocolo ModBus.		
Argumentos	**Tipo**	**Descripción**
SlaveAddr	E	Dirección ModBus RTU del servidor.
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.
-4	S	Error, la librería no fue inicializada.
-20	S	Error, el valor del argumento **Qty** es invalido.
Notas		**Descripción**
1		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbRtuClGetLibStatus() o asincrónicamente desde el evento @OnMbRtuClientRx().
2		Con la función MbRtuClGetRxReg() es posible leer los datos recibidos y con la función MbRtuClGetExceptionCode() puede obtener la excepción ModBus retornada.

Ejemplo 1:

```
// Enviar requerimiento al servidor ModBus RTU (dirección 5) para leer
// 4 entradas discretas a partir de la dirección 10001 de memoria.

if( MbRtuClSendReadDiscretes(5, 10001, 4) < 0 )
{
   // Error.
}
```

MbRtuClSendReadHoldingReg (SlaveAddr, StartAddr, Qty): Envía un requerimiento al servidor para “Read Holding Registers” acorde a la función número 3 del protocolo ModBus.		
Argumentos	**Tipo**	**Descripción**
SlaveAddr	E	Dirección ModBus RTU del servidor.
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.
-4	S	Error, la librería no fue inicializada.
-20	S	Error, el valor del argumento **Qty** es invalido.
Notas		**Descripción**
1		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbRtuClGetLibStatus() o asincrónicamente desde el evento @OnMbRtuClientRx().
2		Con la función MbRtuClGetRxReg() es posible leer los datos recibidos y con la función MbRtuClGetExceptionCode() puede obtener la excepción ModBus retornada.

Ejemplo 1:

```
// Enviar requerimiento al servidor ModBus RTU (dirección 5) para leer
// 6 registros a partir de la dirección 40001 de memoria.

if( MbRtuClSendReadHoldingReg(5, 40001, 6) < 0 )
{
   // Error.
}
```

MbRtuClSendReadInputReg (SlaveAddr, StartAddr, Qty): Envía un requerimiento al servidor para "Read Input Registers" acorde a la función número 4 del protocolo ModBus.		
Argumentos	**Tipo**	**Descripción**
SlaveAddr	E	Dirección ModBus RTU del servidor.
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.
-4	S	Error, la librería no fue inicializada.
-20	S	Error, el valor del argumento **Qty** es invalido.
Notas		**Descripción**
1		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbRtuClGetLibStatus() o asincrónicamente desde el evento @OnMbRtuClientRx().
2		Con la función MbRtuClGetRxReg() es posible leer los datos recibidos y con la función MbRtuClGetExceptionCode() puede obtener la excepción ModBus retornada.

Ejemplo 1:

```
// Enviar requerimiento al servidor ModBus RTU (dirección 5) para leer
// 6 registros a partir de la dirección 40001 de memoria.

if( MbRtuClSendReadInputReg(5, 30001, 6) < 0 )
{
   // Error.
}
```

MbRtuClSendWriteCoil(SlaveAddr, StartAddr, Value): Envía un requerimiento al servidor para "Write Single Coil" acorde a la función número 5 del protocolo ModBus.		
Argumentos	**Tipo**	**Descripción**
SlaveAddr	E	Dirección ModBus RTU del servidor.
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.
-4	S	Error, la librería no fue inicializada.
Notas		**Descripción**
1		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbRtuClGetLibStatus() o asincrónicamente desde el evento @OnMbRtuClientRx().
2		Con la función MbRtuClGetRxReg() es posible leer los datos recibidos y con la función MbRtuClGetExceptionCode() puede obtener la excepción ModBus retornada.

Ejemplo 1:

```
// Enviar requerimiento al servidor ModBus RTU (dirección 5) para activar
// la salida 6.

if( MbRtuClSendWriteCoil(5, 6, 0xFF00) < 0 )
{
   // Error.
}
```

MbRtuClSendWriteReg(SlaveAddr, StartAddr, Value): Envía un requerimiento al servidor para "Write Single Register" acorde a la función número 6 del protocolo ModBus.		
Argumentos	**Tipo**	**Descripción**
SlaveAddr	E	Dirección ModBus RTU del servidor.
Addr	E	Dirección de la salida 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.
-4	S	Error, la librería no fue inicializada.
Notas		**Descripción**
1		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbRtuClGetLibStatus() o asincrónicamente desde el evento @OnMbRtuClientRx().
2		Con la función MbRtuClGetRxReg() es posible leer los datos recibidos y con la función MbRtuClGetExceptionCode() puede obtener la excepción ModBus retornada.

Ejemplo 1:

```
// Enviar requerimiento al servidor ModBus RTU (dirección 5) y escribir
// el valor 666 en el registro 40001 de memoria.

if( MbRtuClSendWriteReg (5, 40001, 666) < 0 )
{
   // Error.
}
```

MbRtuClSendWriteMultCoils(SlaveAddr, StartAddr, Qty, Values[]): Envía un requerimiento al servidor para "Write Multiple Coils" acorde a la función número 15 del protocolo ModBus.		
Argumentos	**Tipo**	**Descripción**
SlaveAddr	E	Dirección ModBus RTU del servidor.
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.
-4	S	Error, la librería no fue inicializada.
-10	S	Error, dirección del array inválida.
-20	S	Error, valor **Qty** inválido.
Notas		**Descripción**
1		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbRtuClGetLibStatus() o asincrónicamente desde el evento @OnMbRtuClientRx().
2		Con la función MbRtuClGetRxReg() es posible leer los datos recibidos y con la función MbRtuClGetExceptionCode() puede obtener la excepción ModBus retornada.

Ejemplo 1:

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

// Enviar requerimiento al servidor ModBus RTU (dirección 5) y escribir
// 16 salidas a partir de la dirección 1 de memoria. Las salidas
// 1 a 12 se activaran con el valor 1, y las salidas 13 a 16
// se desactivaran con el valor 0.

if(MbRtuClSendWriteMultCoils (5, 1, 16, Values) < 0 )
{
   // Error.
}
```

MbRtuClSendWriteMultReg(SlaveAddr, StartAddr, Qty, Values[]): Envía un requerimiento al servidor para "Write Multiple Registers" acorde a la función número 16 del protocolo ModBus.		
Argumentos	**Tipo**	**Descripción**
SlaveAddr	E	Dirección ModBus RTU del servidor.
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.
-4	S	Error, la librería no fue inicializada.
-10	S	Error, dirección del array inválida.
-20	S	Error, valor **Qty** inválido.
Notas		**Descripción**
1		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbRtuClGetLibStatus() o asincrónicamente desde el evento @OnMbRtuClientRx().
2		Con la función MbRtuClGetRxReg() es posible leer los datos recibidos y con la función MbRtuClGetExceptionCode() 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 RTU (dirección 5) y escribir
// 4 registros a partir de la dirección 40001 de memoria. En el
// servidor el registro 40001 tendrá el valor 111,
// el registro 40002 el valor 222, etc.

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

MbRtuClSendReadWriteMultReg(SlaveAddr, 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**
SlaveAddr	E	Dirección ModBus RTU del servidor.
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.
-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		El usuario puede esperar la respuesta del servidor comprobando el estado de la librería con la función MbRtuClGetLibStatus() o asincrónicamente desde el evento @OnMbRtuClientRx().
2		Con la función MbRtuClGetRxReg() es posible leer los datos recibidos y con la función MbRtuClGetExceptionCode() 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 RTU (dirección 5) y leer
// 5 registros a partir de la dirección 40001 de memoria. Al
// mismo tiempo escribir 4 registros a partir de la
// dirección 40001 con los valores del array Values[].

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

7.1.3 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.

MbRtuClGetLibStatus(): 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 3 en página 21 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.
0	S	Respuesta recibida con éxito del servidor, listo.
-4	S	Error, timeout en transacción. Especificado por función de inicialización.
-6	S	Error, ver código de retorno de función Pawn/Ladder.
-7	S	Error, se recibió un código ModBus de excepción.
-10	S	Error, código de función ModBus recibido difiere al enviado.
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:

```
// Direccion del servidor o esclavo ModBus.
#define SLAVE_ADDR   (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 RTU.
// Velocidad 9600 bps, formato 8E1, timeout 10 segundos, interface RS232.

MbRtuClInit(9600, SERIAL_8E1, 10, MB_RTU_INTERFACE_RS232)

for(;;)
{
   // Realizar transaccion ModBus si DIN5 = 1.
   if(DinValue(DIN5) && WaitResponse == 0)
   {
      // Enviar peticion para leer "Holding Registers".
      if(MbRtuClSendReadHoldingReg(SLAVE_ADDR, 42010, 1) < 0)
      {
         // Error en transmision.
      }
      else
      {
         WaitResponse = 1
      }
   }

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

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

        // 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 RTU, que se encuentra en la dirección (1) del protocolo ModBus (ver constante **SLAVE_ADDR**). La función **MbClRtuInit()** especifica la velocidad, formato de transmisión y timeout.

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 **MbClRtuClSendReadHoldingReg()**.

Luego se lee el código de estado de la librería con **MbRtuClGetLibStatus()** y se almacena en la variable **MbStat**. A continuación se comprueba si el código es 0 (respuesta recibida), en caso afirmativo se emplea **MbRtuClGetRxReg()** para obtener los valores de los registros recibidos y copiarlos en la variable **RxData[ ]**. Notar que utilizamos en esta función como último argumento **ByteOffset=1**, ello se debe a que la respuesta ModBus de esta función contiene los registros de 16-bits desplazados un byte, por ende, informamos de este desplazamiento para obtenerlos correctamente (ver especificaciones del protocolo ModBus).

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 **MbRtuClSendReadHoldingReg()** 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 **MbRtuClGetLibStatus()**, y así evitar el uso de valores numéricos.

Tabla 3: Códigos de estados retornado por MbRtuClGetLibStatus()

Nombre	Valor	Descripción
MB_RTU_CL_STAT_SENDING	127	Requerimiento en proceso de envío.
MB_RTU_CL_STAT_LIB_NOT_INIT	126	La librería no fue inicializada aun.
MB_RTU_CL_STAT_RESP_RXED	125	Respuesta recibida, pero no procesada.
MB_RTU_CL_STAT_INITIALIZED	124	La librería fue inicializada.
MB_RTU_CL_STAT_OK	0	Respuesta recibida con éxito del servidor, listo.
MB_RTU_CL_STAT_ERR_TO	-4	Error, timeout en transacción. Especificado por función de inicialización.
MB_RTU_CL_STAT_ERR_FUNC	-6	Error, ver código de retorno de función Pawn/Ladder.
MB_RTU_CL_STAT_ERR_EXCEP	-7	Error, se recibió un código ModBus de excepción.
MB_RTU_CL_STAT_ERR_FUNC_DIF	-10	Error, código de función ModBus recibido difiere al enviado.

Ejemplo:

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

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

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

MbRtuClGetRxReg(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 3 en página 21 para constantes.
2		La dirección ModBus RTU retornada por el servidor, no es devuelta por esta función.

Ejemplo:

```
new RxData[4]

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

MbRtuClGetRxBytes(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 invalido de ByeOffset.
Notas		**Descripción**
1		Antes de leer en buffer, es útil verificar el estado de la librería, ver Tabla 3 en página 21 para constantes.
2		La dirección ModBus RTU retornada por el servidor, no es devuelta por esta función.

Ejemplo:

```
new RxData[4]

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

MbRtuClGetFuncCode(): 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**
1		Debe coincidir con la última petición enviada al servidor.

Ejemplo:

```
new FuncCode

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

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 4: 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

MbRtuClGetExceptionCode(): 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 5: 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.

MbRtuClGetDataLength(): Obtiene la cantidad de bytes de datos 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		No tiene en cuenta campos ADDRESS, FUNCTION, ni CRC.
2		Solo para usos de depuración.

Ejemplo:

```
new Length

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

MbRtuClGetSlaveAddr (): Obtiene la dirección del servidor o esclavo ModBus RTU a la que pertenece el ultimo paquete de datos recibido.		
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		Si se envió requerimiento a una dirección Broadcast, no hay respuesta del servidor o esclavo.
2		Debe coincidir con la última petición enviada al servidor.

Ejemplo:

```
new SlaveAddr

// Obtener dirección de esclavo de ultima respuesta.
SlaveAddr = MbRtuClGetSlaveAddr()
```

7.1.4 Funciones para Eventos

Las siguientes funciones le permiten activar o desactivar eventos.

MbRtuClSetRxEvent(): Activa el evento @OnMbRtuClientRx().		
Argumentos	**Tipo**	**Descripción**
-	-	
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, el evento no pudo ser creado.
Notas		**Descripción**
1		El evento @OnMbRtuClient() se activa cuando una transacción termina y el código de estado de la librería (ver Tabla 3 en página 21 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 @OnMbRtuClientRx(), de esta forma el código principal puede realizar otras tareas.

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

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

   // Inicializar Cliente ModBus RTU.
   MbRtuClInit(9600, SERIAL_8E1, 10, MB_RTU_INTERFACE_RS232)

   // Activar evento "OnMbRtuClientRx" para determinar respuesta
   // del servidor.
   if(MbRtuClSetRxEvent() < 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(MbRtuClSendReadHoldingReg(1, 40001, 4) < 0)
               {
                     LcdClear()
                     LcdPrintf(0,1, "Send error...")

                     WaitResponse = 0
               }
       }
   }

   // Retorno.
   return 0
}

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

@OnMbRtuClientRx()
{
      new MbStat

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

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

      // Leer registros recibidos si no hay error.
      if(MbStat == 0)
      {
         // Leer Registros.
         MbRtuClGetRxReg(RxData, 0, 4, 1)

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

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

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

Ejemplo:

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

8 Abreviaciones y Términos Empleados

- **PLC**: Programable Logic Controller (Controlador Lógico Programable).
- **RTU**: Remote Terminal Unit.
- **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.

9 Historial de Revisiones

Tabla 6: Historia de Revisiones del Documento

Revisión	Cambios	Descripción	Estado
03 27/DIC/2014	1	1. Cambios mínimos, descripción de función MbRtuClInit().	Preliminar
02 29/AUG/2014	2	1. Corrige nombre de funciones: MbRtuClSendWriteReg() y MbRtuClSendWriteMultReg().	Preliminar
01 10/AUG/2014	1	1. Versión preliminar liberada.	Preliminar

10 Referencias

Ninguna.

11 Información Legal

11.1 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.

12 Información de Contacto

Para mayor información, visítenos en www.slicetex.com

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

Slicetex Electronics
Córdoba, Argentina

13 Contenido

13.1 Índice de Tablas