Slicetex Ladder Designer Studio

NOTA DE APLICACIÓN

AN023
ModBus RTU – Servidor (Slave)

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 servidor (Slave) 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.*

***ModBus RTU** permite el empleo del protocolo **ModBus** con dispositivos que se comunican mediante un enlace serial, que habitualmente se realiza por un medio físico tipo RS232 o RS485.*

*Este documento detalla el uso del protocolo **ModBus RTU** en modo servidor (Slave), que le permitirá conectar otros dispositivos clientes (Master) al PLC. De esta manera podrá recibir y enviar datos a otros PLC, paneles HMI o algún software SCADA que operen como cliente ModBus (Master). 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://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 AN023, 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 configurarse como un servidor **ModBus RTU**, aceptando conexiones de clientes ModBus RTU. La comunicación se realiza a través de un enlace serial, que puede provenir de un medio físico RS232, RS485 u otro, dependiendo del hardware del PLC.

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

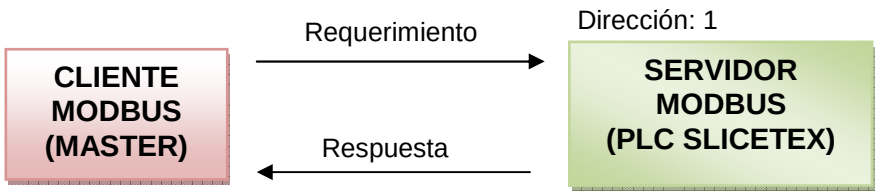


Fig. 1: Transacción ModBus RTU.

La figura 1, muestra el proceso que lleva a cabo el PLC cuando se realiza una transacción ModBus RTU para procesar un requerimiento de un cliente ModBus. Los pasos son los siguientes:

1. **Configuración**: El PLC debe configurar los parámetros para funcionar como servidor ModBus, por ejemplo la dirección ModBus RTU, velocidad del puerto, formato de datos, etc.
2. **Requerimiento**: El cliente 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 cliente recibe la respuesta.
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 servidor:

Tabla 1: Funciones ModBus Soportadas

Función	Código	Descripción
Read Coils	1	Leer estado de salidas discretas (relés o bits de memoria).
Read Discrete Inputs	2	Leer estado de entradas discretas.
Read Holding Register	3	Leer valores de registros “Holding”.
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.

4.2 Excepciones ModBus

Cuando el servidor ModBus detecta un error en la transacción, devuelve un error que corresponde a un Código de Excepción. Las posibles excepciones que puede devolver el servidor, se listan a continuación:

Tabla 2: Excepciones ModBus (Resumen)

Excepción	Código Hexa	Descripción
ILLEGAL FUNCTION	1	Función no soportada por el servidor.
ILLEGAL DATA ADDRESS	2	Dirección invalida.
ILLEGAL DATA VALUE	3	Valor de datos inválidos. También retornado cuando el paquete recibido está incompleto.
SLAVE DEVICE FAILURE	4	Error interno en el servidor.
SLAVE BUSY	6	El servidor no puede aceptar la petición, está ocupado.

5 Particularidades del Servidor ModBus RTU

5.1 Mapa de Direcciones

A continuación se muestra el mapa de direcciones del servidor ModBus RTU implementado:

Tabla 3: Mapa de direcciones por modelo de PLC.

PLC (MODELO)	Uso	Rango de Direcciones
STX8060/8081	Discrete Outputs (Coils) [1]	1-8
STX8060/8081	Discrete Outputs (GP-Coils) [2]	6001-6128
STX8081	Discrete Inputs	10001-10008
STX8060	Discrete Inputs	10001-10010
STX8060/8081	Holding Registers Read / Write [3]	42001-42032

La tabla indica como están asignados los recursos ModBus en los diferentes modelos de PLC que son accesibles por un cliente ModBus.

Por ejemplo, para acceder a los relés del PLC modelo STX8081, hay que enviar un requerimiento con la función ModBus número 1 (**Read Coils**) a un rango valido de direcciones, en este caso las direcciones 1 a 8 reflejan el estado de los 8 reles, en formato 1-bit por salida.

Para leer la entrada discreta número 10 del PLC modelo STX8060, se debe enviar un requerimiento con la función ModBus número 2 (**Read Discrete Inputs**) a la dirección 10010.

Finalmente, los registros “Holding” contienen registros con datos de 16-bits. Dichos datos pueden ser escritos o leídos por el cliente ModBus (a partir de la dirección 42001).

- Nota[1]: Si el PLC funciona con un programa creado en lenguaje Ladder, al escribir sobre un relé desde un cliente, el PLC hará efectivo la escritura en el hardware al final del SCAN CYCLE actual.
- Nota[2]: Se puede leer/escribir un máximo de 32 direcciones al mismo tiempo.
- Nota[3]: Se puede leer/escribir un máximo de 16 direcciones al mismo tiempo.

6 ModBus Servidor con Lenguaje Ladder

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

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

En este documento llamaremos “transacción” al proceso de recibir un requerimiento y enviar una respuesta al cliente ModBus.

En lenguaje Ladder es muy simple configurar el servidor ModBus RTU. Básicamente hay 4 clases de componentes disponibles:

- Funciones para configurar el servidor y la conexión.
- Funciones para cargar registros (que el cliente puede leer).
- Funciones para leer registros (que el cliente ha modificado).
- Funciones para leer y cargar GP-COILS (bits de memoria de uso general).

En la página siguiente repasaremos brevemente como utilizar **ModBus RTU** como servidor en Ladder.

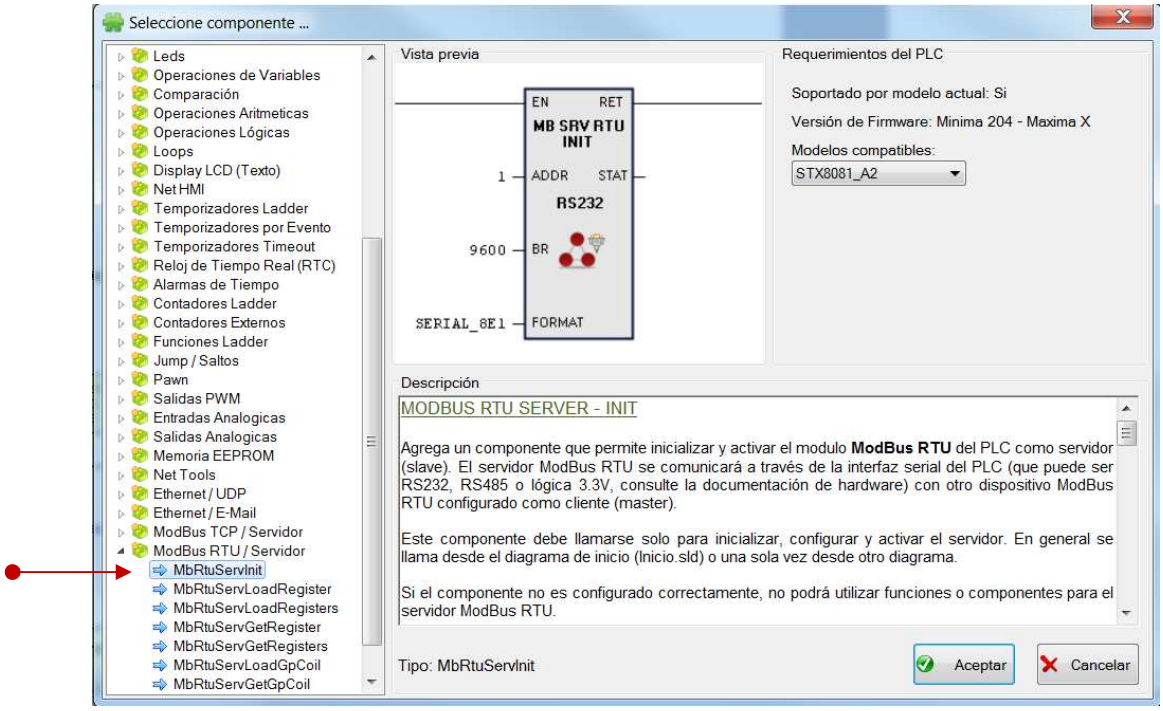
Importante:

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 …**” del menú contextual desplegable del componente.

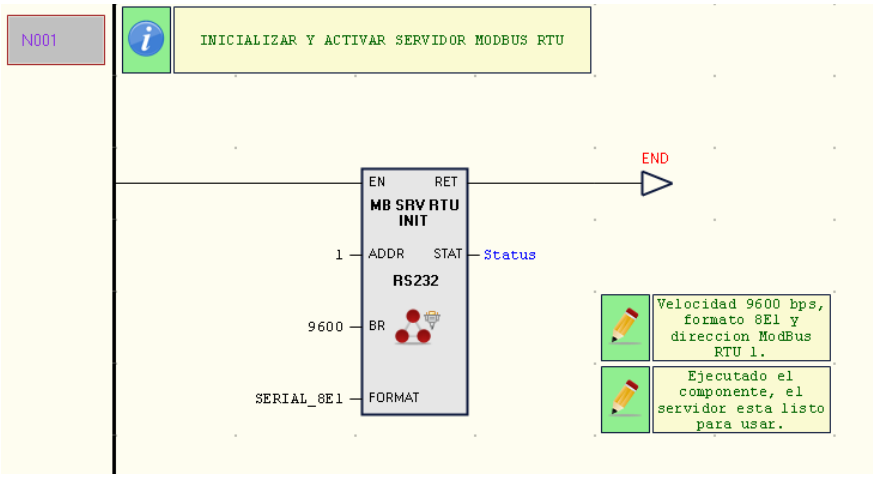
6.1 Componentes Ladder

6.1.1 Configuración

Los componentes Ladder para el servidor ModBus RTU, se encuentran en el selector de componentes del entorno StxLadder. En el selector, navegue a la sección "**ModBus RTU / Servidor**". Luego puede seleccionar el componente **MbRtuServInit**, como se muestra en la imagen a continuación:



Puede insertar el componente **MbRtuServInit** 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 **ADDR**, específica la dirección ModBus RTU que el servidor tendrá dentro de la red ModBus. Es un número entre 1 y 247. El cliente que se conecte a nuestro servidor, debe utilizar esta dirección.

La entrada **BR**, determina el BaudRate o velocidad en bits-por-segundo de la conexión serial. Puede ser cualquier valor, pero se recomiendan utilizar valores estándares, por ejemplo: 9600 (preferentemente) o 19200. Velocidades bajas le permiten al PLC mejorar el rendimiento de procesador para otras aplicaciones.

La entrada **FORMAT**, permite establecer el formato de datos utilizado para la transmisión serie entre los dispositivos. Por ejemplo, la constante "**SERIAL_8E1**" significa 8-bits de datos, paridad Even (par) y 1-bit de stop (valor recomendado por la entidad regulatoria de ModBus, pero es posible utilizar otros valores).

Finalmente, es posible seleccionar la interfaz eléctrica de comunicación desde las propiedades del componente, por ejemplo **RS232**, **RS485**, etc. La misma se mostrará gráficamente luego en el centro del componente. Tenga en cuenta que algunos modelos de PLC o módulos de expansión pueden requerir configuración extra para la interfaz seleccionada. Consulte la hoja de datos del PLC o dispositivo para más información.

Si el componente es ejecutado sin errores, el servidor ModBus RTU está listo para ser accedido por clientes remotos.

Importante: Recuerde que los parámetros de conexión **BR** y **FORMAT**, deben coincidir con los utilizados por el cliente ModBus que se intente conectar con nuestro servidor.

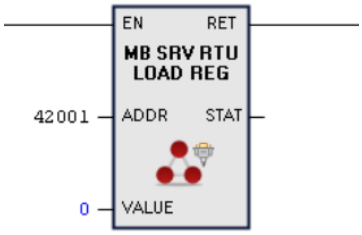
6.1.2 Carga de Registros

Los siguientes componentes permiten cargar los registros "**Holding Registers**" con valores específicos:

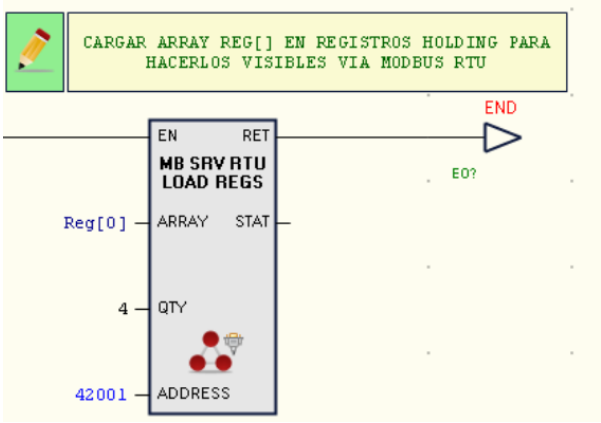
- **MbRtuServLoadRegister**
- **MbRtuServLoadRegisters**

Los registros Holding pueden ser leídos por un cliente que se conecte al servidor.

En el caso del componente **MbServLoadRegister**, la entrada **ADDR** especifica la dirección del registro a escribir en el servidor (en este caso el 42001) con el valor de la entrada **VALUE**. Dicho registro luego puede ser leído por un cliente ModBus RTU.



El componente **MbRtuServLoadRegisters**, permite escribir varios registros al mismo tiempo utilizando un array de datos. En el siguiente ejemplo, la entrada la entrada **ARRAY** se conecta a un array de al menos 4 elementos con valores tipo **Int32**. Luego se especifican la cantidad de elementos a escribir en la entrada **QTY** del componente y en **ADDR** se especifica la dirección del registro inicial a escribir en el servidor (en este caso el 42001).



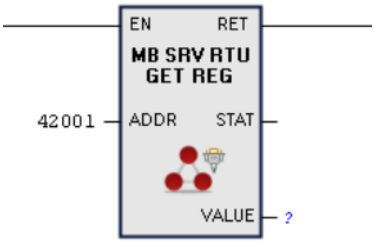
Una vez ejecutado el componente (**EN**=1), la dirección 42001 se escribe con el valor del primer elemento de la variable **Reg**[0] conectada a la entrada **ARRAY**. Luego se escribe la dirección 42002 con el segundo elemento **Reg**[1] y así sucesivamente con los 4 elementos especificados en la entrada **QTY**, hasta llegar al registro 42004.

6.1.3 Lectura de Registros

Los siguientes componentes permiten leer los registros "**Holding Registers**", que es muy útil para recibir los datos escritos en el servidor por un cliente ModBus RTU remoto:

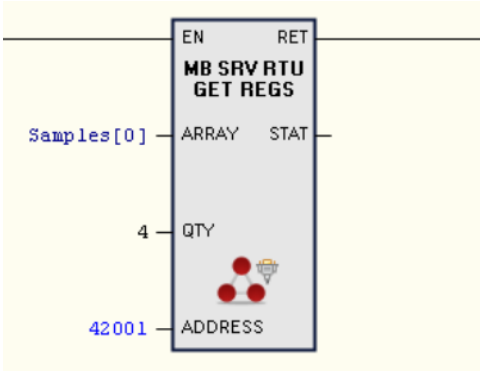
- **MbRtuServGetRegister**
- **MbRtuServGetRegisters**

El componente **MbRtuServGetRegister** devuelve en la salida **VALUE** el valor del registro leído en la dirección especificada por la entrada **ADDR**.



Por ejemplo, si necesitamos leer el registro 42017, especificamos en **ADDR**=42017 y en **VALUE** conectamos una variable del tipo **Int32**. Luego de ejecutarse el componente, obtendremos el valor del registro.

El componente **MbRtuServGetRegisters** devuelve en **ARRAY** el valor de **QTY** registros leídos a partir de la dirección especificada por la entrada **ADDR**. Esto es muy útil para leer varios registros al mismo tiempo.



Ejemplo, si necesitamos leer 10 registros a partir de la dirección 42001, conectamos un array con al menos 10 elementos del tipo **Int32** en la entrada **ARRAY**, especificamos **QTY**=10 y **ADDRESS**=42001. Al ejecutarse el componente (**EN**=1) el array conectado será actualizado con los valores de los 10 registros en cada uno de sus elementos.

6.1.4 Componentes para Manipular GP-COILS

Las GP-COILS son bits de memoria de propósito general (GP=General Purpose), que pueden ser escritas o leídas por un cliente ModBus.

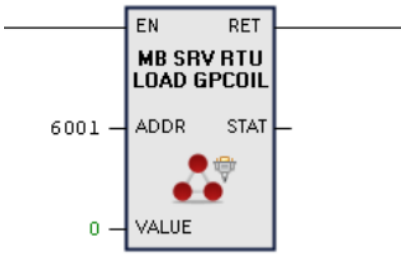
El PLC puede también modificar o leer el valor de las GP-COILS.

Son muy útiles para memoria de uso general en paneles HMI o sistemas SCADA.

A continuación se listan los componentes Ladder para acceder a las GP-COILS:

- **MbRtuServLoadGpCoil**
- **MbRtuServGetGpCoil**

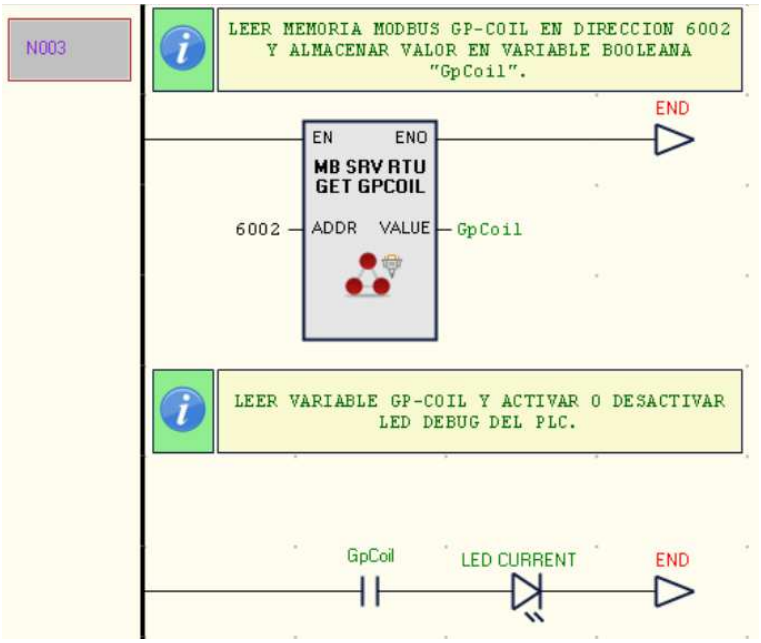
El componente **MbRtuServLoadGpCoil**, permite escribir una GP-COIL en la dirección **ADDR** con el valor del tipo **bool** especificado en la entrada **VALUE**.



Por ejemplo, si escribimos una GP-COIL en la dirección **ADDR**=6001 con el valor **VALUE**=1, un cliente remoto ModBus podrá leer dicho valor y activar una lámpara en un tablero.

El componente **MbRtuServGetGpCoil**, obtiene el valor de una GP-COIL localizada en la dirección **ADDR**, depositando su valor en la variable conectada a la salida **VALUE**.

El siguiente ejemplo, el componente lee una GP-COIL en la dirección 6002 y deposita su valor en la variable **GpCoil**. Luego se comprueba el valor de **GpCoil**, si es "1" se activa el led DEBUG del PLC, de lo contrario se desactiva el led DEBUG. Entonces, un cliente remoto podría activar/desactivar el led DEBUG con tan solo escribir un "1" o un "0" en la dirección 6002 del servidor ModBus.



El ejemplo anterior tiene solo fines didácticos, pero considere otras aplicaciones como: Activación o Desactivación de motores, válvulas, etc de forma remota utilizando las GP-COILS.

7 ModBus Servidor con Lenguaje Pawn

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

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

En este documento llamaremos “transacción” al proceso de recibir un requerimiento y enviar una respuesta al cliente ModBus.

En lenguaje Pawn es muy simple configurar el servidor **ModBus RTU**. Básicamente hay cuatro clases de funciones disponibles:

- Funciones para configurar el servidor y parámetros de la conexión.
- Funciones para cargar registros (que el cliente puede leer).
- Funciones para leer registros (que el cliente ha modificado).
- Funciones para leer y cargar GP-COILS (bits de memoria de uso general).

En la página siguiente se describen dichas funciones en detalle.

7.1 Funciones Nativas en Pawn Disponibles

7.1.1 Funciones de Configuración

MbRtuServInit(Address, Baudrate, FormatMode, Interface): Inicializa y configura los parámetros para que el PLC pueda funcionar como un servidor ModBus RTU.		
Argumentos	**Tipo**	**Descripción**
Address	E	Dirección ModBus RTU del servidor. Un valor entre 1 y 247.
Baudrate	E	Velocidad en bit-por-segundos de la conexión. Debe coincidir con la velocidad del cliente. 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**".
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 servidor ya se encuentra inicializado.
-2	S	Error, falla en inicialización, memoria insuficiente.
-3	S	Error, no se puede crear tarea.
-4	S	Error, falla al inicializar hardware serial.
-5	S	Error, Baudrate inválido.
-6	S	Error, Address inválido.
Notas		**Descripción**
1		El servidor es activado luego de llamar exitosamente a esta función.
2		**Tip**: Al utilizar velocidades de **Baudrate** bajas, por ej. 9600 mejora el desempeño para otras actividades del procesador del PLC. Ya que las peticiones ModBus repetitivas estarán más espaciadas en el tiempo y el procesador podrá emplear su tiempo para otras actividades.
3		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:

```
// Inicializar Servidor ModBus RTU.
// Velocidad 9600 bps, formato 8E1 y dirección ModBus RTU 1.
// Interfaz de comunicacion: RS-232

if(MbRtuServInit(1, 9600, SERIAL_8E1, MB_RTU_INTERFACE_RS232) < 0)
{
      // Error, pausar programa en este punto.
      while(true)
      {
            DelayMS(2000)
            LedToggle()
      }
}
```

7.1.2 Funciones para Cargar Registros

Las siguientes funciones permiten cargar los registros "**Holding Registers**" con valores específicos, dichos registros luego pueden ser leídos por un cliente que se conecte al servidor.

MbRtuServLoadRegisters(StartAddr, Qty, Values[ ]): Carga con valores los registros Holding.		
Argumentos	**Tipo**	**Descripción**
StartAddr	E	Dirección del primer registro a cargar. Se utiliza el mismo valor que ve el cliente.
Qty	E	Cantidad de registros a cargar. Máximo depende de cantidad de registros disponibles.
Values[]	E	Array con los valores a escribir en los registros. De cada elemento del array, solo se toman los 16 bits menos significativos, es decir solo 2 bytes por elemento.
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, librería no inicializada.
-4	S	Error, dirección de array inválida.
-10	S	Error, argumento **Qty** invalido.
-11	S	Error, dirección **StartAddr** inválida.
-12	S	Error, dirección **StartAddr** invalidad para el valor **Qty**.
Notas		**Descripción**
-		

Ejemplo 1:

```
        // Leer canal analogico 1.
        Samples[0] = VinRead(1)

        // Leer canal analogico 2.
        Samples[1] = VinRead(2)

        // Leer canal analogico 3.
        Samples[2] = VinRead(3)

        // Leer canal analogico 4.
        Samples[3] = VinRead(4)

        //
        // Copiar array "Samples" con las cuatro muestras de los
        // canales analógicos en los "Holding Registers"
        // a partir de la dirección 42001. De esta forma
        // serán visibles por un cliente ModBus RTU.
        //

        MbRtuServLoadRegisters(42001, 4, Samples)
```

MbRtuServLoadRegister (StartAddr, Value]): Carga un valor en un registro Holding.		
Argumentos	**Tipo**	**Descripción**
StartAddr	E	Dirección del registro a cargar. Se utiliza el mismo valor que ve el cliente.
Value	E	Valor a escribir en el registro. Solo se toman los 16 bits menos significativos, es decir solo 2 bytes.
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, librería no inicializada.
-11	S	Error, dirección **StartAddr** inválida.
Notas		**Descripción**
-		

Ejemplo 1:

```
//
// Cargar el valor 666 en el registro 42001.
//

MbRtuServLoadRegister(42001, 666)
```

7.1.3 Funciones para Leer Registros

Las siguientes funciones permiten leer los registros "**Holding Registers**", dichos registros pueden ser escritos por un cliente que se conecte al servidor.

MbRtuServGetRegisters(StartAddr, Qty, Data[ ], Index): Obtiene los valores de los registros Holding.		
Argumentos	**Tipo**	**Descripción**
StartAddr	E	Dirección del primer registro a leer. Se utiliza el mismo valor que ve el cliente.
Qty	E	Cantidad de registros a leer. Máximo depende de cantidad de registros disponibles.
Values[]	S	Array donde se copiaran los valores de los registros leídos. En cada elemento del array se copia un registro de 16-bits.
Index	E	Índice donde comiénzala copia en el array.
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, librería no inicializada.
-4	S	Error, dirección de array inválida.
-10	S	Error, argumento **Qty** invalido.
-11	S	Error, dirección **StartAddr** inválida.
-12	S	Error, dirección **StartAddr** invalidad para el valor **Qty**.
Notas		**Descripción**
-		

Ejemplo 1:

```
  new Values[4]

  // Obtener el valor de 4 registros a partir de la dirección
  // 42001 y almacenarlos a partir del elemento 0 en el array Values[].

  MbRtuServGetRegisters(42001, 4, Values, 0)
```

MbRtuServGetRegister(StartAddr, &Value): Obtiene el valor de un registro Holding.		
Argumentos	**Tipo**	**Descripción**
StartAddr	E	Dirección del registro a leer. Se utiliza el mismo valor que ve el cliente.
Value	S	Valor del registro leído. Solo se toman los 16 bits menos significativos, es decir solo 2 bytes.
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, librería no inicializada.
-4	S	Error, dirección de **Value** es invalida.
-11	S	Error, dirección **StartAddr** inválida.
Notas		**Descripción**
-		

Ejemplo 1:

```
new RegValue = 0

// Leer registro 42001 y almacenar en RegValue.
MbRtuServGetRegister(42001, RegValue)
```

7.1.4 Funciones para Manipular GP-COILS

Las GP-COILS son bits de memoria de propósito general (GP=General Purpose), que pueden ser escritas o leídas por un cliente ModBus.

El PLC puede también modificar o leer el valor de las GP-COILS.

Son muy útiles para memoria de uso general en paneles HMI o sistemas SCADA.

A continuación se listan las funciones Pawn para acceder a las GP-COILS.

MbRtuServLoadGpCoil(Address, Value): Escribe un valor en una GP-COIL.		
Argumentos	**Tipo**	**Descripción**
Address	E	Dirección de la GP-COIL a escribir.
Value	E	Valor a escribir (1 o 0).
Retorno	**Tipo**	**Descripción**
0	S	Operación exitosa.
-1	S	Error, librería no inicializada.
-11	S	Error, dirección **StartAddr** inválida.
Notas		**Descripción**
-		

Ejemplo 1:

A continuación se escriben dos GP-COILS con valores 0 y 1. Luego un cliente ModBus (panel HMI o software SCADA) podrá leer dichos valores en las direcciones correspondientes (6002 y 6032) mediante la función “Read Coils (1)” del protocolo (ver **Tabla 1** en página 4).

```
// Escribir un 0 en GP-COIL en dirección 6002.
MbRtuServLoadGpCoil(6002, 0)

// Escribir un 1 en GP-COIL en dirección 6032.
MbRtuServLoadGpCoil(6032, 1)
```

MbRtuServGetGpCoil(Address): Lee el valor de una GP-COIL.		
Argumentos	**Tipo**	**Descripción**
Address	E	Dirección de la GP-COIL a leer.
Retorno	**Tipo**	**Descripción**
0	S	El valor de la GP-COIL es 0. Ver nota[1] también.
1	S	El valor de la GP-COIL es 1.
Notas		**Descripción**
1		Si la librería ModBus RTU no fue inicializada o la dirección de la GP-COIL es invalida, el valor retornado es 0 también.

Ejemplo 1:

A continuación, se lee la GP-COIL en dirección 6002. Si su valor es “1” se activa el led-debug. Si es “0” se desactiva el led-debug. Un cliente/master ModBus RTU puede alterar este valor con la función 5 o 15 del protocolo (ver **Tabla 1** en página 4).

```
// Activar/Desactivar Led Debug de acuerdo a valor del
// GP-COIL en dirección 6002.
if(MbRtuServGetGpCoil(6002))
{
      LedOn()
}
else
{
      LedOff()
}
```

En la práctica, podemos suponer que un panel HMI puede estar alterando este valor para indicar alguna condición al PLC.

8 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.
- **ModBus TCP**: Protocolo ModBus adaptado a la red Internet, en general mediante Ethernet.
- **ModBus RTU**: Protocolo ModBus que utiliza un medio serial (RS232 o RS485) para transferencia de datos.

9 Historial de Revisiones

Tabla 4: Historia de Revisiones del Documento

Revisión	Cambios	Descripción	Estado
03 27/DEC/2014	1	1. Se actualiza configuración para interfaz RS232, RS485 en Ladder. Ver pág. 7. 2. Se actualiza configuración para interfaz RS232, RS485 en Pawn. Ver pág. 13,	Preliminar
02 24/APR/2013	1	1. Se agrega información para utilizar ModBus con lenguaje Ladder, ver página 6.	Preliminar
01 06/APR/2013	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 y consultas, envíe un mail a: info@slicetex.com

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

Slicetex Electronics
Córdoba, Argentina

13 Contenido

13.1 Índice de Tablas