

# Driver Modicon Modbus

Nome do Arquivo	Modbus.dll
Fabricante	Modicon
Equipamentos	Qualquer equipamento compatível com o protocolo Modbus v1.1b
Protocolo	Modbus v1.1b
Versão	4.0.4
Última Atualização	22/06/2020
Plataforma	Win32 e Windows CE (Pocket PC ARM, HPC2000 ARM, HPC2000 X86 e HPC2000 MIPS)
Dependências	Nenhuma
Leitura com Superblocos	Sim
Nível	0

## Introdução

Este Driver implementa o protocolo Modbus, permitindo a uma aplicação da **Elipse Software** comunicar com qualquer equipamento escravo que implemente este protocolo nos modos **ASCII**, **RTU** ou **TCP**.

Este Driver atua sempre como mestre de uma rede Modbus. Se houver necessidade de comunicar com dispositivos mestre, é necessário utilizar o Driver Modbus Slave da **Elipse Software**, que pode ser baixado no *site da empresa*.

O Driver Modbus, a partir da versão 2.00, passou a ser desenvolvido com a biblioteca **IOMKit** da **Elipse Software**. Esta biblioteca é responsável por implementar o acesso ao meio físico (**Serial**, **Ethernet**, **Modem** ou **RAS**). Para obter informações sobre a configuração do **IOMKit**, consulte o tópico **Documentação das Interfaces de Comunicação**.

Recomenda-se iniciar a leitura pelo tópico **Guia de Configuração Rápida** caso o equipamento seja perfeitamente aderente ao protocolo Modbus padrão, definido pela *Organização Modbus (modbus.org)*, e se houver apenas a necessidade de ler ou escrever bits e registros, não necessitando dos recursos mais avançados do Driver.

Para o entendimento mais aprofundado de todas as funcionalidades do Driver recomenda-se iniciar lendo, nesta ordem, os capítulos **Adicionando o Driver em uma Aplicação Elipse** e **Configuração**.

Para a criação de aplicações de grande porte, recomenda-se também a leitura do tópico **Dicas de Otimização**.

Caso não conheça o protocolo, consulte os seguintes tópicos:

- **O Protocolo Modbus**
- **Sites Recomendados**
- **Funções Suportadas**
- **Funções Especiais**

# Guia de Configuração Rápida

Este tópico descreve os passos necessários para configurar o Driver Modbus para a comunicação com equipamentos aderentes ao protocolo padrão definido pela *Organização Modbus*, considerando as opções de configuração mais comuns.

Se o equipamento é perfeitamente aderente ao protocolo padrão, e se deseja-se apenas ler ou escrever registros ou bits, os três tópicos a seguir provavelmente são suficientes para a configuração do Driver:

- **Inserindo o Driver**
- **Configurando o Driver**
- **Configurando Tags de Comunicação**

## Inserindo o Driver

Se estiver utilizando o **Eclipse E3** ou o **Eclipse Power**, leia o tópico **Adicionando o Driver em uma Aplicação Eclipse - Eclipse E3 e Eclipse Power**.

Se estiver utilizando o **Eclipse SCADA**, leia o tópico **Adicionando o Driver em uma Aplicação Eclipse - Eclipse SCADA**.

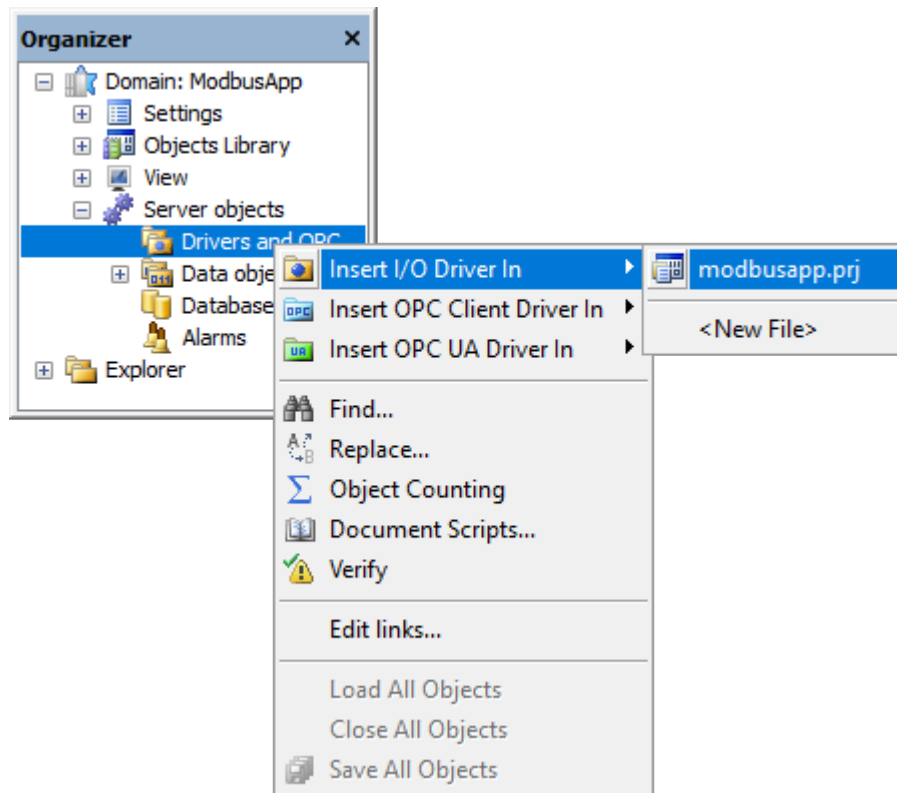
Depois, leia o **passo seguinte** deste Manual, que ensina como configurar o Driver em sua janela de configuração para os casos mais comuns.

## Adicionando o Driver em uma Aplicação da Elipse Software

Esta seção descreve como adicionar o Driver Modbus em aplicações **E3** ou **Eclipse Power** e **Eclipse SCADA**.

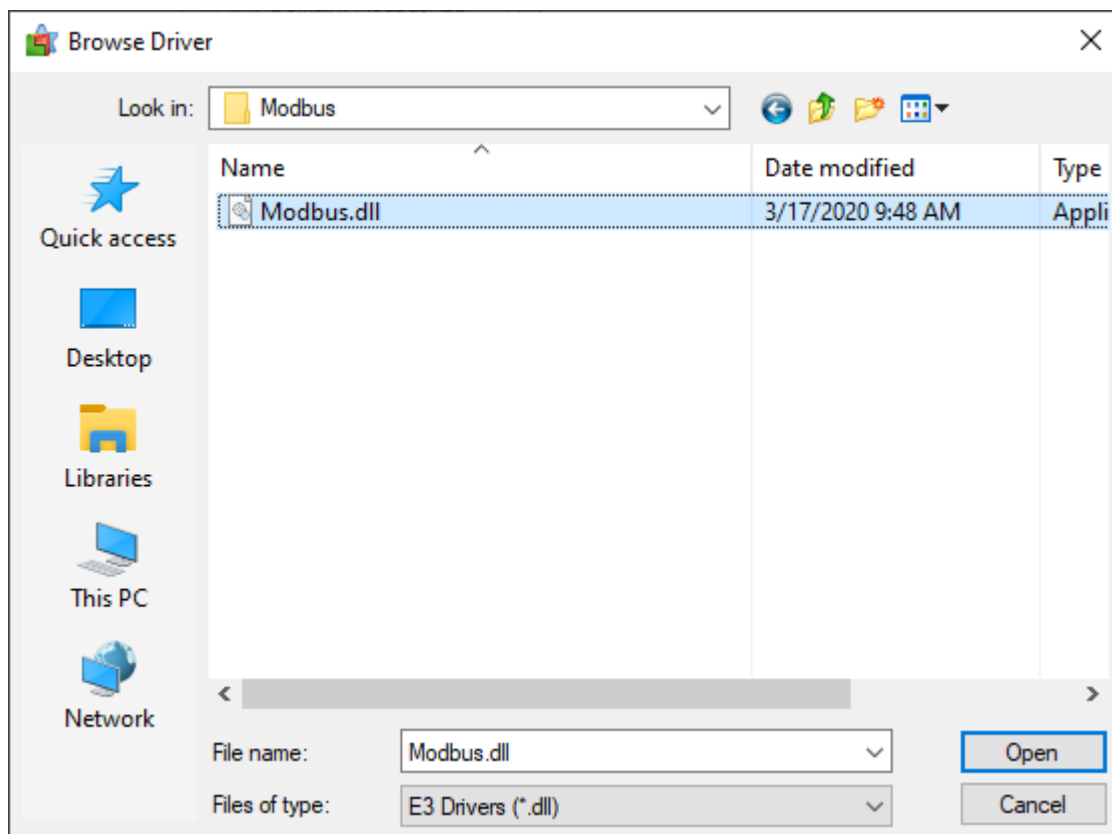
## E3 ou Elipse Power

No Organizer, clique com o botão direito do mouse no item **Server Objects - Drivers and OPC** (*Objetos de Servidor - Drivers e OPC*), selecione a opção **Insert I/O Driver In** (*Inserir Driver de Comunicação em*) e selecione o projeto desejado.



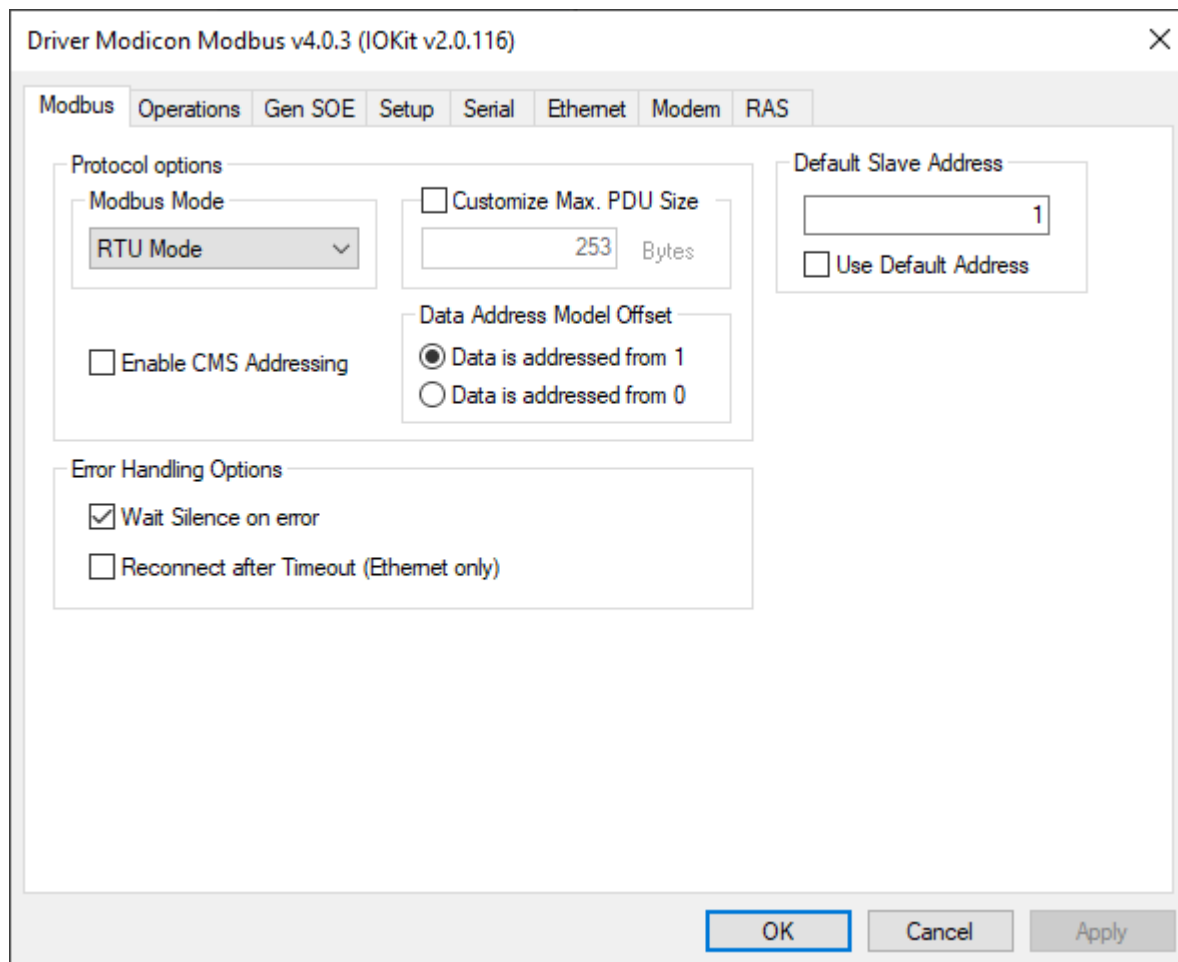
### Adicionar um Driver a uma aplicação E3 ou Elipse Power

Na janela que se abre, selecione o Driver (o arquivo deve ser descompactado em uma pasta no computador em uso) e clique em **Open** (*Abrir*).




### Selecionando um Driver

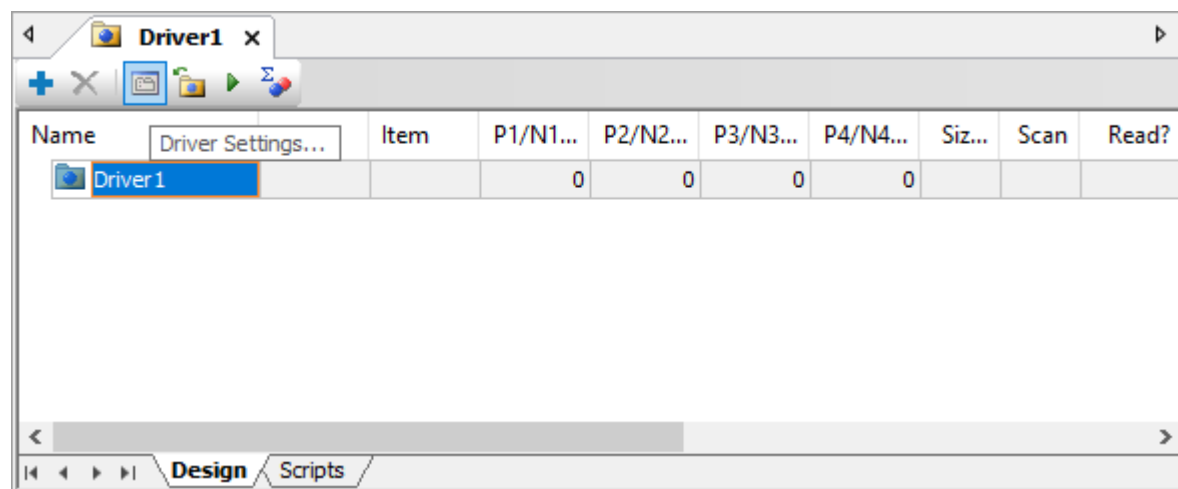
A janela de configurações do Driver abre-se automaticamente, conforme mostrado na figura a seguir.



Janela de configurações do Driver

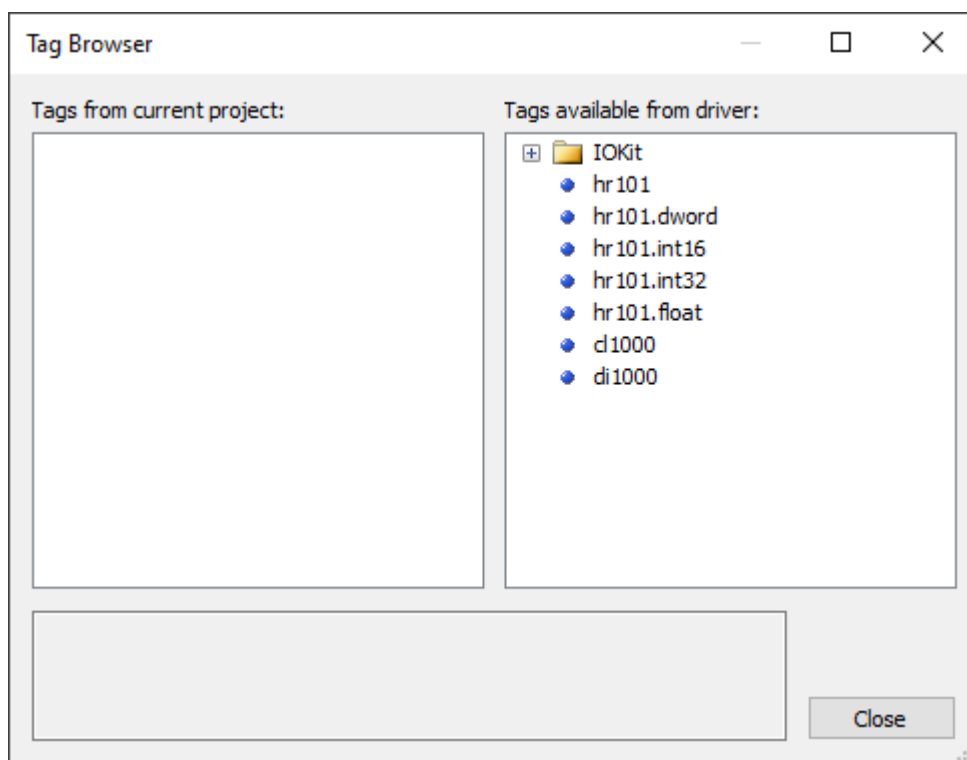
No **segundo passo** do tópico **Guia de Configuração Rápida** é apresentado o procedimento para a configuração básica do Driver, para os casos de uso mais comuns. No tópico **Propriedades** a configuração é apresentada em detalhes.

A janela de configuração do Driver pode sempre ser aberta, posteriormente, clicando-se em **Driver settings** (Configurar o driver) , conforme mostrado na figura a seguir.




Opção Driver settings (Configurar o driver)

Após a configuração das propriedades do Driver, clique em **OK** para que se abra a janela do Tag Browser, permitindo inserir na aplicação Tags pré-definidos, com base nas configurações mais utilizadas. A figura a seguir mostra a janela do Tag Browser. Para adicionar Tags, arraste-os da lista **Tags available from driver** (Tags disponibilizados pelo driver) para a lista **Tags from current project** (Tags do projeto corrente).

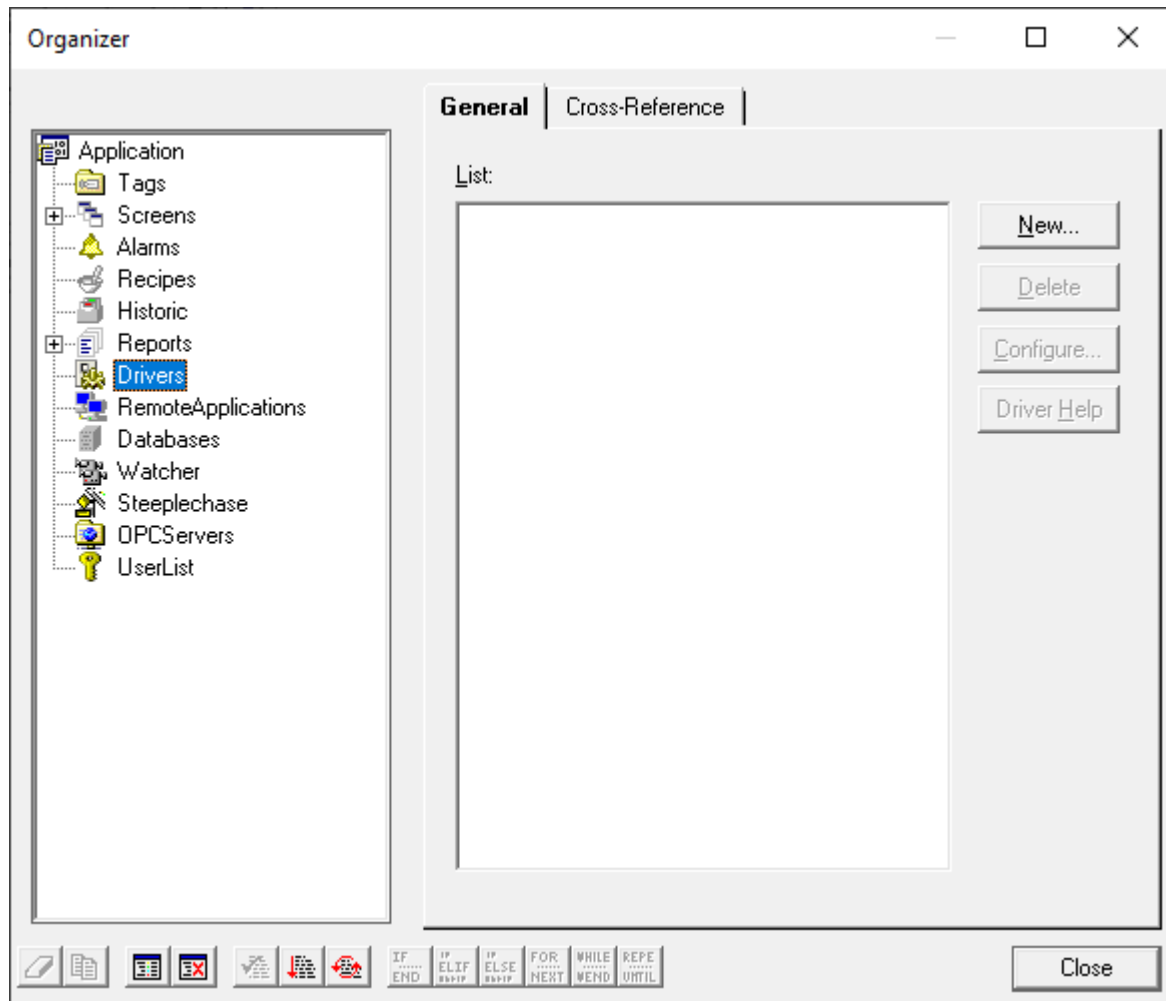


**Janela Tag Browser**

Os Tags disponíveis no Tag Browser são Tags **configurados usando Strings**, novo método que não utiliza o antigo conceito de operações. Insira os mais adequados na aplicação, editando os campos conforme a necessidade. A janela do Tag Browser pode ser aberta posteriormente clicando-se em **Tag Browser** .

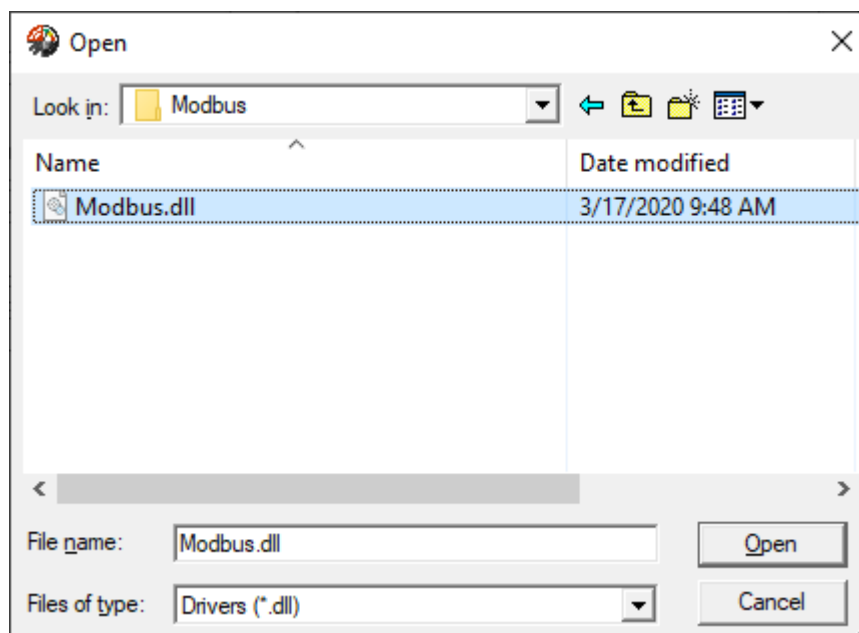
## Elipse SCADA

Através do Organizer, selecione o item **Drivers** e clique em **New** (Novo).



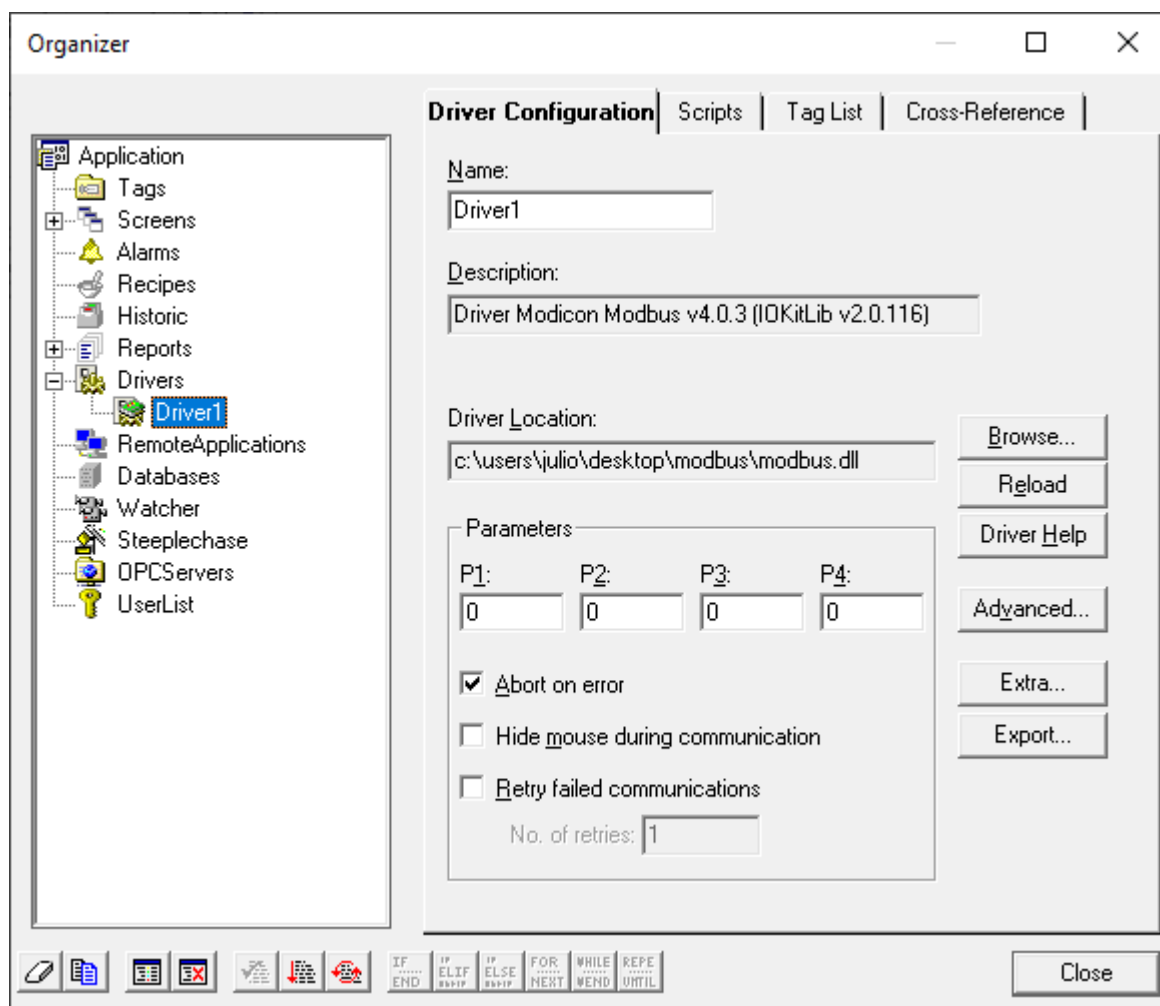
**Adicionar novo Driver ao Elipse SCADA**

Na janela que se abre, selecione o Driver (o arquivo deve ser descompactado em uma pasta no computador em uso) e clique em **Open** (Abrir).



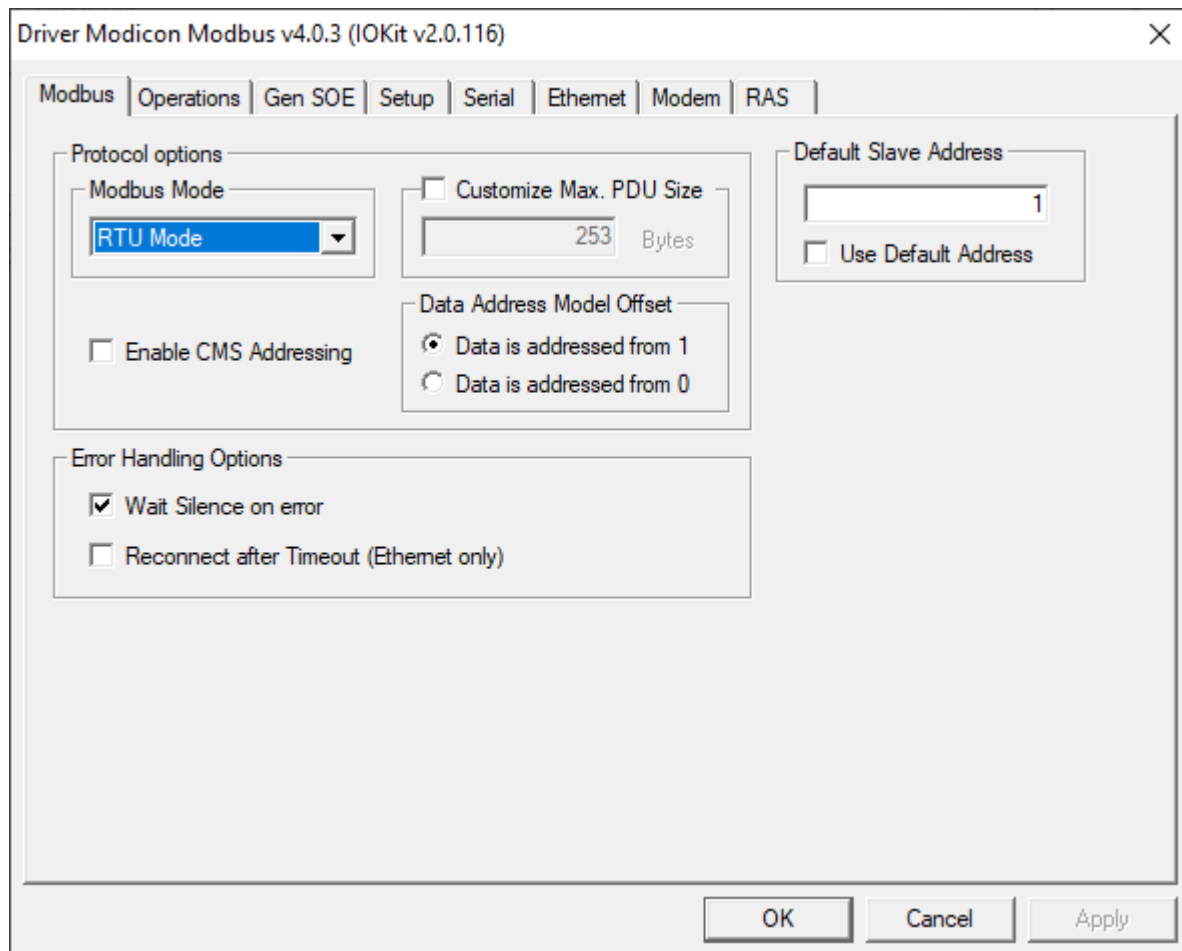
**Selecionando um Driver**

O Driver é então adicionado na aplicação.



**Driver no Organizer**

Para que o Driver funcione corretamente, ainda é preciso configurá-lo em sua janela de configuração. Para abrir esta janela, mostrada na figura a seguir, clique em **Extra** (Extras).



**Janela de configuração do Driver**

O **segundo passo** do **Guia de Configuração Rápida** mostra como configurar o Driver para os casos de uso mais comuns, para equipamentos que aderem aos requisitos do protocolo Modbus padrão. No tópico **Propriedades** a configuração é descrita em detalhes, incluindo os recursos avançados de configuração.



# Configurando o Driver

Após inserir o Driver na aplicação, deve-se abrir a sua janela de configurações, conforme já explicado nos tópicos **Elipse E3 ou Power** ou **Elipse SCADA**. Com a janela de configuração aberta, siga estes passos:

- Configure a camada física de comunicação:
  - Na aba **Setup**, selecione a camada física (**Serial**, **Ethernet**, **Modem** ou **RAS**) a ser utilizada na conexão com o equipamento.
  - Configure a camada física selecionada na aba correspondente (**Serial**, **Ethernet**, **Modem** ou **RAS**).
  - Se precisar de mais informações sobre a configuração da camada física, consulte o tópico **Documentação das Interfaces de Comunicação**.
- Na aba **Modbus**, selecione o modo do protocolo (**RTU**, **ASCII** ou **TCP**) utilizado pelo equipamento. Como regra geral, deve-se selecionar **RTU** ou **ASC** (para a maioria dos equipamentos é **RTU**) para meio físico **Serial** ou **Modem**, ou **TCP** para meio físico **Ethernet** ou **RAS**. As demais opções em geral podem ser deixadas com suas configurações padrão. Caso precise de mais informações sobre as opções desta aba, consulte o tópico **Aba Modbus**.

## NOTA

Em novas aplicações, recomenda-se fortemente evitar o uso de **ModbusRTU** (modo **RTU**) encapsulado em meio **Ethernet TCP/IP**. Entretanto, se por algum motivo, para aplicações legadas, for necessário usar **ModbusRTU** encapsulado em **TCP/IP**, não deixe de habilitar a opção **Reconnect after Timeout**, descrita no tópico **Aba Modbus**.

- Caso se esteja criando a aplicação em produtos mais recentes da Elipse Software como o **E3**, **Elipse Power** ou **Elipse OPC Server**, é possível utilizar a configuração de Tags por **Strings** (campos **Dispositivo** e **Item**). Neste caso, vá para o **passo seguinte** deste guia.
- Caso precise ainda utilizar a antiga configuração numérica (parâmetros *N/B*), usada no **Elipse SCADA**, é importante examinar a aba **Operations**. Observe as sete operações padrão já pré-configuradas no Driver. As operações são configurações de funções e formatações de dados que são posteriormente referenciadas pelos Tags da aplicação. Estas sete operações padrão, já disponíveis quando o Driver é aberto pela primeira vez, são as mais comumente necessárias. Avalie as funções de leitura e escrita e o tipo de dados usado por cada operação e verifique quais delas são necessárias para a aplicação. Caso as operações pré-definidas não se enquadrem nas necessidades, é necessário editá-las ou mesmo criar novas operações. Se for este o caso, leia o tópico **Aba Operations**. A tabela a seguir lista as sete operações pré-definidas.

### Operações pré-definidas

OPERAÇÃO	FUNÇÃO DE LEITURA	FUNÇÃO DE ESCRITA	TIPO DE DADOS	FINALIDADE
1	3: Read Holding Registers	16: Write Multiple Registers	Word	Leitura e escrita de inteiros de 16 bits sem sinal
2	3: Read Holding Registers	16: Write Multiple Registers	DWord	Leitura e escrita de inteiros de 32 bits sem sinal
3	3: Read Holding Registers	16: Write Multiple Registers	Int16	Leitura e escrita de inteiros de 16 bits com sinal
4	3: Read Holding Registers	16: Write Multiple Registers	Int32	Leitura e escrita de inteiros de 32 bits com sinal

OPERAÇÃO	FUNÇÃO DE LEITURA	FUNÇÃO DE ESCRITA	TIPO DE DADOS	FINALIDADE
<b>5</b>	3: Read Holding Registers	16: Write Multiple Registers	<b>Float</b>	Leitura de valores com ponto flutuante de 32 bits
<b>6</b>	1: Read Multiple Coils	15: Write Multiple Coils	<b>Bit</b>	Leitura e escrita de bits
<b>7</b>	2: Read Discrete Inputs	None	<b>Bit</b>	Leitura de bits de um bloco de dados de Entradas Discretas ( <i>Discrete Inputs</i> )

#### NOTA

As sete operações padrão estão configuradas partindo do princípio que o equipamento segue o ordenamento de bytes (*byte order*) padrão do protocolo Modbus, o padrão *big endian*, no qual os bytes mais significativos vêm antes. Se o equipamento não segue este padrão, consulte o tópico **Aba Operations** para informações sobre como configurar operações para diferentes ordenamentos de bytes.

Para informações detalhadas sobre a configuração do Driver, leia o tópico **Configuração**.

O **passo seguinte** demonstra como configurar Tags de Comunicação com base nas operações pré-definidas.

# Configurando Tags de Comunicação


A seguir são descritas a configuração dos Tags de Comunicação no **E3, Elipse Power** e no antigo **Elipse SCADA** para os usos mais comuns.

## Configuração de Tags no E3 e no Elipse Power


A configuração de Tags no **E3** e no **Elipse Power** pode ser realizada pelo novo método de **configuração por Strings** ou pelo antigo método de **configuração numérica**, compatível com o **Elipse SCADA**. Para novos projetos, recomenda-se usar a configuração por **Strings**, que melhora a legibilidade da aplicação e facilita a manutenção.

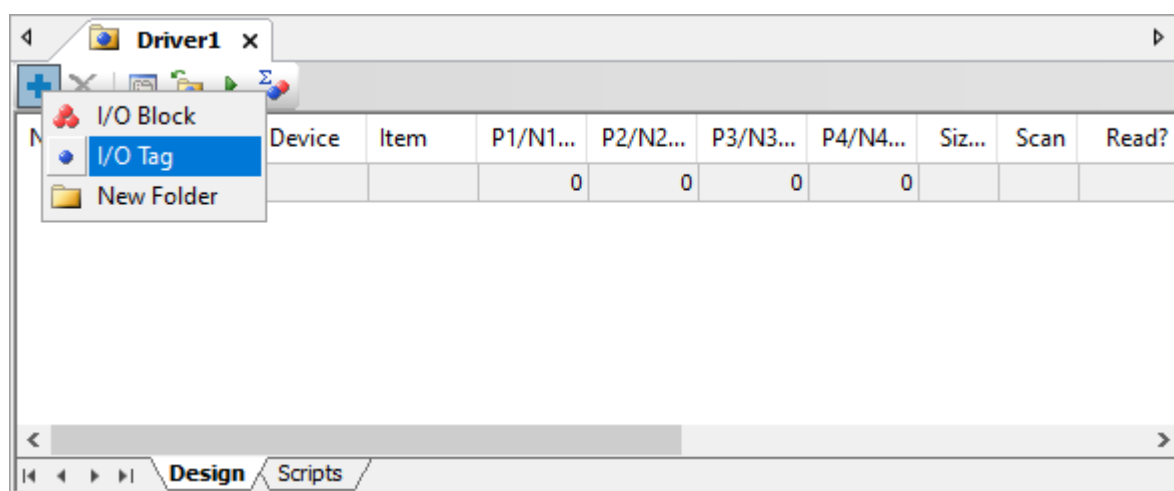
A seguir estão descritos os procedimentos recomendados para a configuração por **Strings** e também para a antiga configuração numérica, caso seja necessária para aplicações legadas.

## Configuração por Strings

Para adicionar Tags configurados por **Strings**, o usuário tem a opção de importar modelos pré-definidos do Tag Browser, conforme explicado no tópico **Adicionando o Driver em uma Aplicação da Elipse Software**. Para isto, deve-se manter a opção **Show Operations in Tag Browser** desabilitada na **aba Operations**, e abrir o Tag Browser clicando em **Tag Browser** .

Para adicionar um novo Tag à aplicação sem o Tag Browser, siga estes passos:

1. No Organizer, clique duas vezes no Driver, selecione a aba **Design**, clique em **Add (Adicionar)**  e selecione o item **I/O Tag (Tag de Comunicação)**, conforme a figura a seguir.



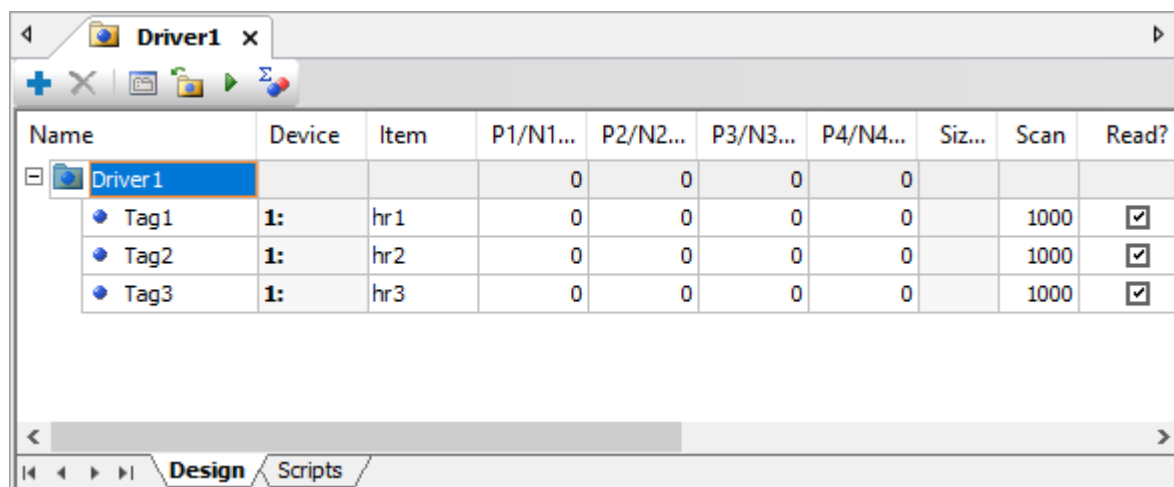
Adicionar novo Tag de Comunicação

2. Na janela **Adding IOTag (Inserindo IOTag)**, configure o campo **Quantity (Quantity)** com o valor 1 (um) e especifique um nome para o Tag no campo **Name (Nome)**. Clique em **OK** para criar o novo Tag.
3. Na coluna **Device (Dispositivo)**, digite o valor numérico do *Slave Id* do equipamento a comunicar, seguido de dois pontos, como por exemplo "1:" para um *Slave Id* igual a 1 (um). Note que, em um meio **Ethernet TCP/IP**, muitas vezes este valor é ignorado, sendo utilizado apenas o endereço IP e a porta configurada na aba **Ethernet**, que deve constar na documentação do equipamento.
4. Na coluna **Item**, especifique o mnemônico do **espaço de endereçamento** (conjunto de funções Modbus de leitura e escrita) seguido do endereço do registro ou bit. Para *Holding Registers*, o espaço de endereçamento é "hr" ou "shr", onde este último não permite escrita em blocos, pois usa a função de escrita **06 (Write Single Register)**, enquanto o espaço de endereçamento "hr" usa a função de escrita **16 (Write Multiple Registers)**. Ambos usam a função Modbus de leitura **03 (Read Holding Registers)**. Para *Coils* utilize "cl" ou "scl". Novamente, a diferença é que este último, que usa a função **05 (Force Single Coil)**, não escreve em blocos. A seguir são fornecidos alguns exemplos de configuração da coluna **Item**.

- a. Leitura ou escrita do *Holding Register* 150, usando as funções **03** e **16** (escrita de múltiplos registros): **Item** deve ser igual a "hr150".
  - b. Leitura ou escrita do *Holding Register* 150, usando as funções **03** e **06** (escrita de registros simples): **Item** deve ser igual a "shr150".
  - c. Leitura ou escrita do *Coil* de endereço FFF0h (65520), usando as funções **01** e **15** (escrita de múltiplos bits): **Item** deve ser igual a "cl65520" ou "cl&hFFF0" (o prefixo "&h" pode ser usado para fornecer endereços em formato hexadecimal).
  - d. Leitura ou escrita do *Coil* de endereço FFF0h (65520) usando as funções **01** e **05** (escrita de bits simples, um a um): **Item** deve ser igual a "scl65520" ou "scl&hFFF0" (o prefixo "&h" pode ser usado para fornecer endereços em formato hexadecimal).
5. Para mais informações sobre os demais recursos da configuração por **Strings**, como outras funções Modbus, funções especiais e diferentes tipos de dados, consulte o tópico **Configuração por Strings**.
  6. O endereçamento dos Tags deve corresponder ao mapa de endereços Modbus do equipamento, que deve constar na documentação do fabricante. Em caso de dúvida, consulte o tópico **Dicas de Endereçamento**.

Configure preferencialmente Tags simples (chamados de Tags PLC no antigo **Elipse SCADA**) ao invés de Tags Bloco, mantendo o recurso de Superblocos habilitado (propriedade **EnableReadGrouping** configurada como Verdadeiro), deixando à aplicação e ao Driver a otimização do agrupamento. Para mais detalhes, veja o tópico **Leitura por Superblocos**.

A título de exemplo, a figura a seguir mostra Tags configurados por **Strings**.



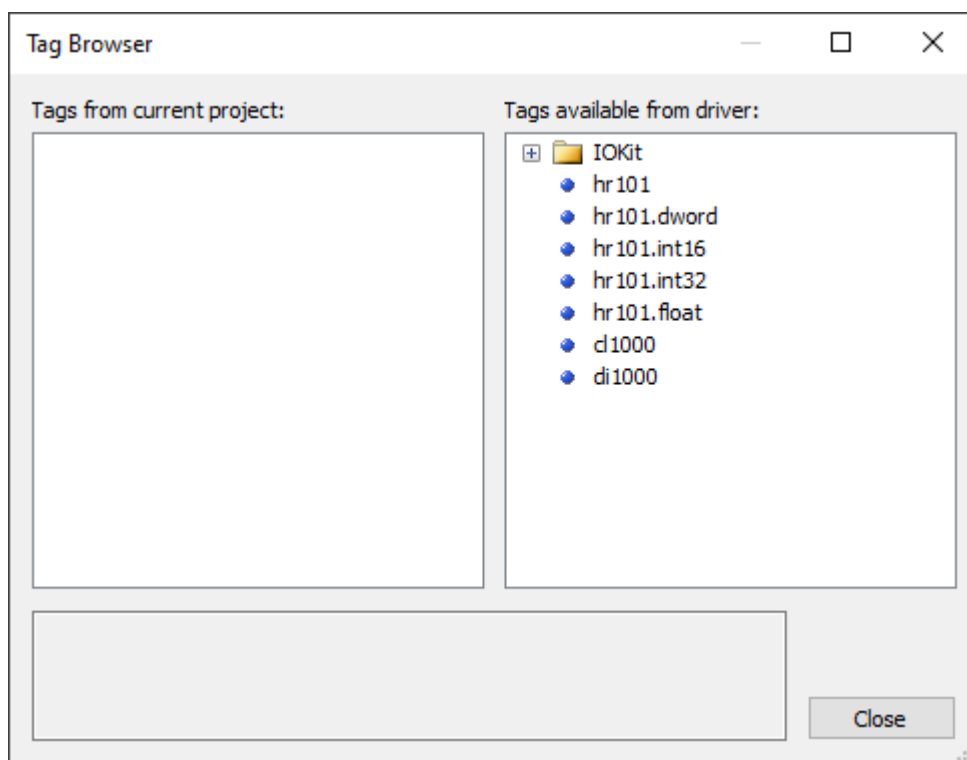
Name	Device	Item	P1/N1...	P2/N2...	P3/N3...	P4/N4...	Siz...	Scan	Read?
Driver1			0	0	0	0			
Tag1	1:	hr1	0	0	0	0		1000	<input checked="" type="checkbox"/>
Tag2	1:	hr2	0	0	0	0		1000	<input checked="" type="checkbox"/>
Tag3	1:	hr3	0	0	0	0		1000	<input checked="" type="checkbox"/>

Exemplo de Tags configurados por Strings

## Configuração Numérica

No caso do **E3** ou do **Elipse Power**, é possível usar o Tag Browser para criar Tags com as operações pré-definidas, **configuradas numericamente**. Para isto, selecione a opção **Show Operations in Tag Browser** na aba **Operations**.

A janela do Tag Browser, mostrada na figura a seguir, é aberta ao clicar-se em **OK** ao fechar a janela de configurações do Driver.



**Tag Browser para a configuração numérica de Tags**

Para adicionar um novo Tag à aplicação, siga estes passos:

1. Arraste os Tags da lista **Tags available from driver** (*Tags disponibilizados pelo driver*) para a lista **Tags from current project** (*Tags do projeto corrente*), conforme descrito no tópico **E3 ou Elipse Power**. Para muitos equipamentos, a operação **1**, a mais comum, já deve ser suficiente, bastando arrastar para a lista dos Tags de projeto o Tag **Op1<word>**. Note que, supondo que sejam necessários vários Tags com a mesma operação, o que geralmente é o caso, pode-se arrastar o mesmo Tag diversas vezes (perceba que a aplicação acrescenta números sequenciais ao nome padrão). Pode-se também acrescentar um Tag de cada operação e criar cópias mais tarde, no Organizer.
2. Feche o Tag Browser e configure o parâmetro *N4/B4* de cada Tag com o endereço de registro ou bit a ser lido ou escrito, conforme o mapa de registradores do equipamento. Este mapa de endereços deve constar na documentação do fabricante. Em caso de dúvida, consulte o tópico **Dicas de Endereçamento**.
3. Configure também o parâmetro *N1/B1* de cada Tag com o endereço do equipamento (*Slave Id*) a ser acessado em cada caso. Este parâmetro em geral é configurável no equipamento e, para determiná-lo, consulte a documentação ou suporte do fabricante, em caso de dúvida.
4. Renomeie os Tags, se desejar, com um nome que seja mais significativo para a aplicação.

Configure preferencialmente Tags simples (chamados de Tags PLC no antigo **Elipse SCADA**) ao invés de Tags Bloco, mantendo o recurso de Superblocos habilitado (propriedade **EnableReadGrouping** configurada como Verdadeiro), deixando à aplicação e ao Driver a otimização do agrupamento. Para mais detalhes, veja o tópico **Leitura por Superblocos**.

## Configuração de Tags no Elipse SCADA

O **Elipse SCADA** não possui suporte ao Tag Browser, portanto é necessário configurar manualmente os Tags de Comunicação. Deve-se criar Tags com a seguinte configuração:

- **N1/B1:** Endereço do Equipamento (*Slave Id*)
- **N2/B2:** Código da operação

- **N3/B3:** Não usado, deixar em 0 (zero)
- **N4/B4:** Endereço do registro Modbus ou bit

Note que, para este Driver, os parâmetros *N* dos Tags simples têm o mesmo significado dos parâmetros *B* dos Tags Bloco, e por isto são descritos em conjunto.

Em caso de dúvida sobre qual valor atribuir ao parâmetro *N4/B4*, consulte o tópico **Dicas de Endereçamento**.

Uma vez que o **Eclipse SCADA** não possui suporte a Superblocos, recomenda-se priorizar a criação de Tags Bloco, agrupando registros adjacentes ou próximos, de forma a ler o máximo de registros no menor número de requisições do protocolo.

Note também que, uma vez que o equipamento suporte os **limites padrão do protocolo para o tamanho do frame de comunicação**, devido ao recurso de **Partição Automática de Blocos**, não é preciso se preocupar em exceder o tamanho máximo de bloco que o protocolo suporta, pois o Driver já cria as subdivisões apropriadas no momento da comunicação.

## Considerações Finais

Se tudo o que se precisa é utilizar as operações padrão do Driver, e se o equipamento segue o protocolo padrão definido pela Organização Modbus, os três passos apresentados neste Guia de Configuração Rápida devem ser suficientes para configurar o Driver.

Para aplicações de maior porte, recomenda-se também ler o tópico **Dicas de Otimização**.

Maiores detalhes sobre a configuração de Tags de Comunicação são fornecidos no tópico **Configurando um Tag de Comunicação**.

# O Protocolo Modbus

O **Protocolo Modbus** foi desenvolvido inicialmente pela Modicon em 1979, sendo hoje um padrão aberto, mantido pela Organização Modbus (modbus.org), tendo sido implementado por centenas de fabricantes em milhares de equipamentos. A Schneider Electric, atual controladora da Modicon, transferiu os direitos do protocolo para a Organização Modbus em abril de 2004, firmando o compromisso de manter o protocolo aberto. A especificação pode ser obtida gratuitamente no site da Organização ([www.modbus.org](http://www.modbus.org)), e a utilização do protocolo é livre de taxas de licenciamento.

O protocolo é baseado em mensagens de comando e resposta, posicionado no nível 7 do modelo OSI (camada de aplicação), que possibilita comunicação cliente e servidor entre equipamentos conectados a diferentes tipos de redes. Oferece serviços com funções definidas por um código de oito bits. Existem três categorias de códigos de funções:

- **Códigos de funções públicas:** Funções bem definidas pelo protocolo, com garantia de unicidade, validadas pela comunidade Modbus e publicamente documentadas em MB IETF RFC. Podem assumir valores de **1 a 64**, de **73 a 99** e de **111 a 127**
- **Códigos de funções definidas pelo usuário:** Funções não padronizadas, que não precisam de aprovação da Modbus.org, sem qualquer garantia de unicidade, podendo ser livremente implementadas. Podem assumir valores nas faixas de **65 a 72** e de **100 a 110**
- **Códigos de funções reservadas:** Códigos com valores dentro da faixa de funções públicas, atualmente usados por alguns fabricantes em produtos antigos, e não mais disponíveis para uso público. São exemplos os códigos **9, 10, 13, 14, 41, 42, 90, 91, 125, 126 e 127**. Para mais informações, consulte o **Anexo A** da especificação do protocolo (versão 1.1b), que está disponível no *site oficial do protocolo*

Este Driver implementa 11 das 19 funções públicas previstas na versão atual (1.1b) da especificação do protocolo, bem como algumas funções específicas de fabricantes ou relacionadas a recursos específicos do Driver, denominadas **Funções Especiais**. As funções públicas implementadas são descritas no tópico **Funções Suportadas**. As seguintes funções públicas do protocolo ainda não são suportadas:

- **Função 08:** Diagnostic
- **Função 11:** Get Com event counter
- **Função 12:** Get Com Event Log
- **Função 17:** Report Slave ID
- **Função 22:** Mask Write Register
- **Função 23:** Read/Write Multiple Registers
- **Função 24:** Read FIFO queue
- **Função 43:** Read Device Identification

Caso identifique a necessidade de implementar alguma destas funções, entre em contato com o *departamento comercial da Elipse Software*.

## Sites Recomendados

O Driver Modicon Modbus está disponível para *download*, sem custos, no site da **Elipse Software**, na área de *download de Drivers*.

Maiores informações referentes ao protocolo Modbus podem ser obtidas em [www.modbus.org](http://www.modbus.org), site oficial do protocolo.

O **Elipse Modbus Simulator** está disponível para *download* (sem custos) no site da Elipse Software, na área de *download do E3*.

O Simulador Modbus Slave **Modsim**, provavelmente o mais conhecido da categoria, pode ser adquirido em [www.wintech.com/html/modsim32.htm](http://www.wintech.com/html/modsim32.htm). Este software emula o equipamento, permitindo a comunicação com o Driver.

Existe também a alternativa gratuita **Free Modbus PLC Simulator**, disponível para download no site [www.plcsimulator.org](http://www.plcsimulator.org).

Outras alternativas de simuladores e outras ferramentas de software relacionadas ao protocolo podem ser encontradas no *site oficial do protocolo*.

## Funções Suportadas

As funções do protocolo Modbus suportadas por este Driver estão descritas a seguir.

### Funções de Leitura

- **01:** Leitura de **Bit** (*Read Coil Status - 0x*)
- **02:** Leitura de **Bit** (*Read Input Status - 1x*)
- **03:** Leitura de **Words** (*Read Holding Registers - 4x*)
- **04:** Leitura de **Words** (*Read Input Registers - 3x*)
- **07:** Leitura de Status (*Read Exception Status*)
- **20:** Leitura de Registro de Arquivo (*Read File Register - 6x*)

### Funções de Escrita

- **05:** Escrita de **Bit** (*Force Single Coil - 0x*)
- **06:** Escrita de **Word** Simples (*Preset Single Register - 4x*)
- **15:** Escrita de **Bits** (*Force Multiple Coils - 0x*)
- **16:** Escrita de **Words** (*Preset Multiple Registers - 4x*)
- **21:** Escrita de Registro de Arquivo (*Write File Register - 6x*)

Informações detalhadas sobre cada uma destas funções podem ser obtidas na especificação do protocolo Modbus, disponível para *download* no site da *Organização Modbus*.

Além das funções padrão do protocolo, como já referido, este Driver também implementa funções especiais, não definidas pelo protocolo, em geral relacionadas à leitura da memória de massa. A lista das funções especiais suportadas pode ser conferida no tópico **Funções Especiais**. A configuração completa do Driver está descrita no tópico **Configuração**.

Caso identifique a necessidade de adicionar suporte a alguma função nova neste Driver, entre em contato com o *departamento comercial da Elipse Software*.



## Funções Especiais

As funções especiais de leitura e escrita são funções deste Driver que não são definidas pelo protocolo Modbus padrão. Foram desenvolvidas para atender particularidades exclusivas de determinados equipamentos, ou também para disponibilizar, de forma padronizada pelo Driver, recursos não disponíveis no protocolo padrão. O Driver Modbus, na atual versão, inclui as seguintes funções especiais:

### Funções de Leitura

- **65 03:** Leitura da Memória de Massa (*ABB MGE 144*), vista em maiores detalhes no tópico **Leitura de Registros da Memória de Massa de Medidores ABB MGE 144**
- **GE SOE:** Leitura de eventos (*GE PAC RX7 Systems*), vista em maiores detalhes no tópico **Leitura de Buffer de Eventos em Controladores GE PAC RX7**
- **SP SOE:** Leitura de eventos (*Relés Schneider Electric da série SEPAM*), vista em maiores detalhes no tópico **Leitura de Eventos de Relés Schneider Electric SEPAM 20, 40 e 80**
- **GenSOE:** Leitura de SOE com algoritmo genérico, implementado pelo software residente no equipamento escravo (CLP), vista em maiores detalhes no tópico **Algoritmo de Leitura de SOE Genérico da Elipse Software**

### Funções de Escrita

- **65 01:** Reinicializa (executa a operação de *reset*) o medidor de energia (*ABB MGE 144*). Este comando é enviado como um comando simples de escrita (**Write**) do Tag. O campo **Valor** do Tag é ignorado pelo Driver e pode ser deixado em 0 (zero). Para mais informações, consulte o manual do equipamento
- **65 02:** Zera a memória de máximos e mínimos (*ABB MGE 144*). Este comando é enviado como um comando simples de escrita (**Write**) do Tag. O campo **Valor** do Tag é ignorado pelo Driver e pode ser deixado em 0 (zero). Para mais informações, consulte o manual do equipamento

Note que as funções especiais neste Driver, com exceção da função de escrita **65 01**, estão relacionadas direta ou indiretamente à leitura de registros de memória de massa dos respectivos equipamentos. Para mais informações, consulte o tópico **Leitura de Memória de Massa**. Para a descrição da configuração de operações e Tags usando estas funções, leia o tópico **Configuração**.

## Configuração

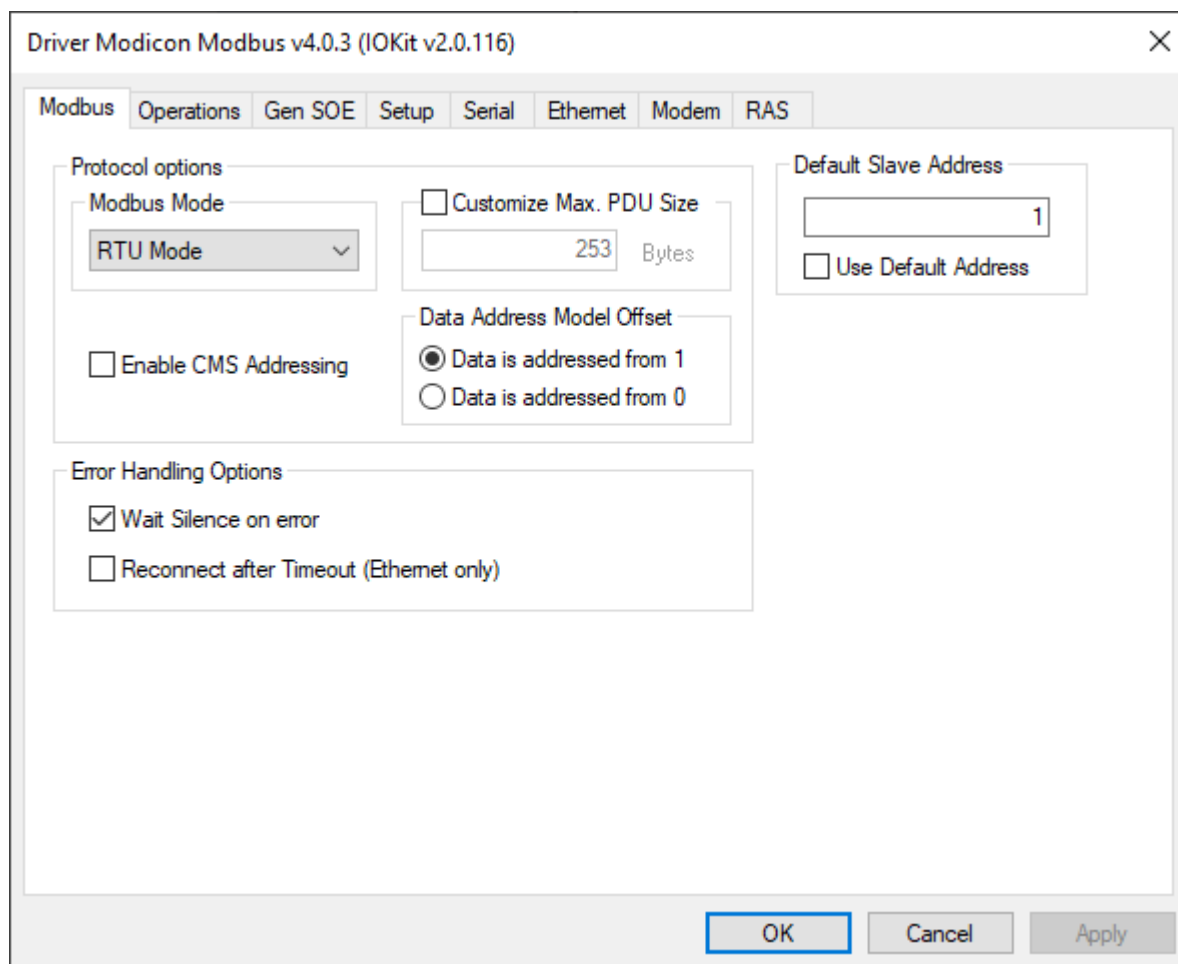
Esta seção descreve como configurar o Driver Modbus. São abordados os seguintes tópicos:

- **Propriedades**
- **Configurando Tags**
- **Leitura de Memória de Massa**

## Propriedades

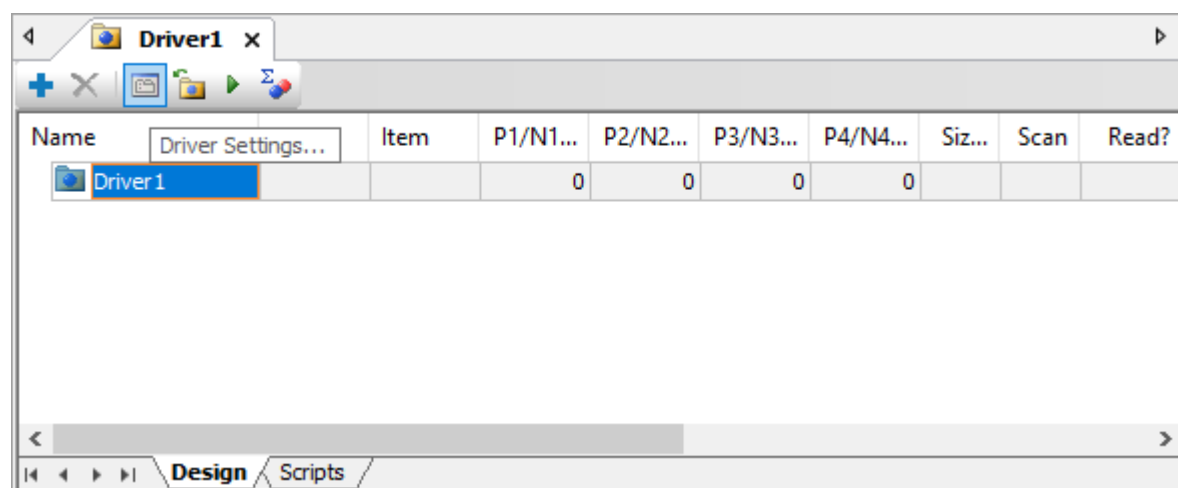
As propriedades do Driver podem ser configuradas em tempo de configuração (*design time*) ou em tempo de execução (*runtime*). A configuração em tempo de execução (*runtime*) também é chamada de **Configuração em Modo Offline** e é descrita em um tópico específico.

Em tempo de configuração, o Driver pode ser configurado por meio de sua janela de configuração, mostrada na figura a seguir.



**Janela de configuração do Driver**

Para abrir a janela de configuração do Driver no **E3** ou no **Elipse Power**, clique duas vezes no objeto Driver no Organizer e clique em **Driver settings** (*Configurar o driver*) , conforme mostrado na figura a seguir.



**Opção Configurar o driver**

Já no **Elipse SCADA**, a janela de configuração do Driver pode ser aberta clicando-se em **Extras**, no Organizador da aplicação.

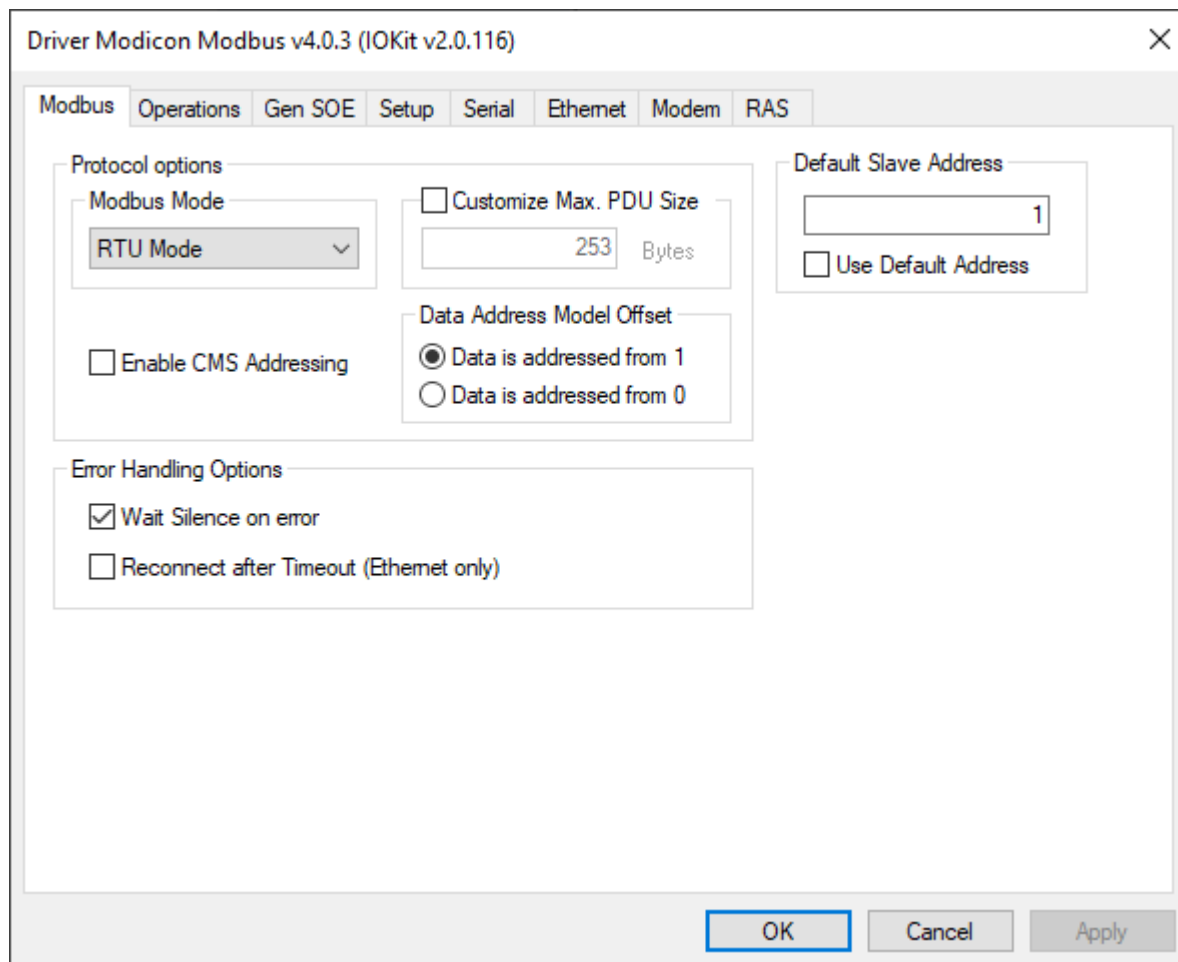
A janela de configuração é dividida em várias abas, algumas para a configuração do **IOLite** e outras específicas do Driver. No caso do Driver Modbus, as abas **Modbus**, **Operations** e **Gen SOE** são específicas. As demais abas são para configuração do **IOLite** e não são descritas neste tópico. Para mais informações sobre a configuração do **IOLite**, consulte o tópico **Documentação das Interfaces de Comunicação**.

Os tópicos a seguir descrevem as abas específicas do Driver e também a configuração em tempo de execução, no chamado **Modo Offline**, utilizando scripts.

- **Aba Modbus**
- **Aba Operations**
- **Aba Gen SOE**
- **Configuração em Modo Offline**

## Aba Modbus

A aba **Modbus** permite a configuração de parâmetros do Driver e do protocolo, conforme a figura a seguir.



**Aba Modbus da janela de configurações do Driver**

As seções a seguir descrevem as opções de configuração presentes nesta aba.

### Protocol Options

Este grupo de opções reúne as opções referentes às variações no padrão do protocolo, conforme a tabela a seguir.

#### Opções de protocolo disponíveis na aba Modbus

OPÇÃO	DESCRIÇÃO
<b>Modbus Mode</b>	<p>Nessa caixa de seleção é possível selecionar o modo do protocolo a ser utilizado. Os modos do protocolo são variações definidas pela norma para melhor adaptá-lo a diferentes meios físicos (<b>Serial</b>, <b>Ethernet TCP/IP</b>, <b>RAS</b>, etc.). São três as opções disponíveis:</p> <ul style="list-style-type: none"> <li>• <b>Modo RTU</b>: Modo padrão para uso em comunicação serial. Inclui CRC de 16 bits</li> <li>• <b>Modo ASCII</b>: Também usado em comunicação serial, é utilizado em equipamentos mais simples, que não suportam os requisitos do modo <b>RTU</b>. Utiliza caracteres ASCII para transmissão, onde cada byte contém dois caracteres ASCII (um por <i>nibble</i>), e por isto menos eficiente que o modo <b>RTU</b> e mais raramente encontrado no mercado. Usa LRC (<i>Longitudinal Redundancy Checking</i>) para a verificação de erros</li> <li>• <b>Modo ModbusTCP</b>: Usado para comunicação em modo <b>TCP/IP</b>. Inclui um campo para verificação de transações e não possui sistema de verificação de erros. O campo de transação permite descartar respostas atrasadas, evitando assim que o Driver assuma como resposta válida para o comando atual os <i>frames</i> de resposta a comandos anteriores. Esta situação pode ocorrer se os modos anteriores forem encapsulados em <b>TCP/IP</b></li> </ul>
<b>Customize Max. PDU Size</b>	<p>Se habilitada, esta opção permite definir um tamanho máximo personalizado para o PDU (<i>Protocol Data Unit</i>). O PDU é a parte do protocolo que não varia entre os modos (<b>ModbusTCP</b>, <b>ASCII</b> e <b>RTU</b>) e que contém a área de dados. O número de bytes de dados suportados em cada comunicação é dado por este valor menos os bytes do cabeçalho, que dependem da função Modbus utilizada. Se esta opção está desabilitada, o tamanho máximo considerado é o <b>valor padrão definido pelo protocolo Modbus versão 1.1b</b>, de 253 bytes. Esta é a opção recomendada para a maioria dos equipamentos</p>
<b>Enable CMS Addressing</b>	<p>Esta opção deve ser usada apenas em equipamentos que suportem o protocolo TeleBUS. Se habilitada, o Driver passa a aceitar um <b>Word</b> de 16 bits como endereço do escravo, ou seja, passa a ser possível definir valores acima de 255 e abaixo de 65535 como endereço do escravo. Neste caso, o endereço do escravo passa a ser definido no protocolo por três bytes. Além disto, a opção <b>Default Slave Address</b> passa a não funcionar mais</p>

OPÇÃO	DESCRIÇÃO
<b>Data Address Model Offset</b>	<p>Esta opção habilita ou desabilita o <i>offset</i> de dados padrão do protocolo, de uma unidade. As opções disponíveis são as seguintes:</p> <ul style="list-style-type: none"> <li>• <b>Data is addressed from 1 (padrão):</b> O endereço fornecido (endereço do campo <b>Item</b> na configuração por <b>Strings</b> ou o parâmetro <i>N4/B4</i> na configuração numérica) é decrementado em 1 (um) antes de ser enviado ao equipamento. Este <i>offset</i> é previsto na especificação do protocolo, e portanto esta é a opção padrão</li> <li>• <b>Data is addressed from 0:</b> O endereço fornecido pelo usuário é usado nas requisições do protocolo, sem alterações</li> </ul> <p>Como regra geral, selecione a primeira opção caso o mapa de registradores do equipamento inicie em 1 (um) e a segunda caso inicie em 0 (zero). Verifique também se o fabricante usa <i>offsets</i> adicionais da antiga <b>Modbus Convention</b>. Para mais informações, consulte a <b>seção a seguir</b></p>

## DICA

Evite usar o modo **RTU** do protocolo encapsulado em meio **Ethernet TCP/IP**. Caso seja necessário encapsular a comunicação serial de equipamentos que utilizem o **Modbus RTU** em **TCP/IP**, existem *gateways* disponíveis no mercado que não somente encapsulam a comunicação serial em **Ethernet TCP/IP**, como também convertem o **Modbus RTU** em **Modbus TCP**. Em último caso, se for inevitável a utilização de **Modbus RTU** em meio **Ethernet TCP/IP**, não deixe de habilitar a opção **Reconnect after Timeout**, descrita na tabela a seguir.

## Data Address Model Offset

Esta opção de configuração, descrita na **tabela anterior**, é fonte de frequentes dúvidas no endereçamento dos Tags de Comunicação, pois há muitas variações na maneira como é implementada pelos fabricantes. A seguir apresentamos mais informações sobre este endereçamento.

No modelo de dados padrão do protocolo são definidos quatro blocos de dados (ou espaços de endereçamento): *Discrete Inputs*, *Coils*, *Input Registers* e *Holding Registers*. Em cada um destes blocos os elementos de dados são endereçados iniciando em 1 (um). Por outro lado, a especificação do *frame* de comunicação define um PDU contendo endereços que podem variar entre 0 (zero) e 65535. A relação entre o endereço fornecido no PDU e o endereço dos elementos de dados, portanto, possui um deslocamento (*offset*) de 1 (um), ou seja, se no PDU de uma requisição constar o endereço 0 (zero), o elemento de dado acessado é o endereço 1 (um).

Com esta opção da aba **Modbus**, o usuário pode escolher se deseja que o Driver ajuste o valor automaticamente, de forma a permitir o uso do endereço do elemento de dado nos Tags (opção padrão), ou se deseja que o valor enviado no PDU seja o mesmo valor fornecido na configuração dos Tags (parâmetro *N4/B4* na **configuração numérica**). Existem equipamentos que seguem o padrão Modbus em seus mapas de endereços (iniciando em um) e outros que mapeiam seus dados sem o *offset* padrão, usando diretamente o valor de endereço presente no *frame* de comunicação (iniciando em zero).

Além deste *offset* unitário, existem ainda equipamentos que utilizam o antigo padrão de *offsets* utilizado pela Modicon, empresa criadora do protocolo, padrão conhecido como **Modbus Convention**, detalhado no tópico **Dicas de Endereçamento**. Consulte no manual do equipamento o mapa de registradores para verificar o padrão utilizado. Em caso de dúvida, consulte o suporte do fabricante.

## NOTA

A opção **Data Address Model Offset** denominava-se **Use Older Address** nas versões anteriores à versão 2.03, onde a opção **Data is addressed from 1** equivale à antiga opção **Use Older Address** habilitada, e a opção **Data is addressed from 0** equivale à opção **Use Older Address** desabilitada.

## Outras Opções

A tabela a seguir descreve as demais opções desta aba, referentes ao comportamento do Driver.

### Outras opções disponíveis na aba Modbus

OPÇÃO	DESCRIÇÃO
<b>Default Slave Address</b>	Este recurso permite configurar um endereço padrão de escravos para que não seja necessário configurá-los em cada Tag. Para utilizar este recurso, configure o <i>Slave Id</i> (parâmetro <i>N1/B1</i> na <b>configuração numérica</b> ou o campo <b>Dispositivo</b> na <b>configuração por Strings</b> ) em 1000, ou seja, todos os Tags com <i>Slave Id</i> igual a 1000 têm este valor substituído pelo valor configurado na caixa de edição <b>Default Slave Address</b> . Também é possível forçar o uso do endereço padrão em todos os Tags, independente do valor de <i>Slave Id</i> configurado, selecionando-se a opção <b>Use Default Address</b>
<b>Wait Silence on error</b>	Se esta opção estiver habilitada, após cada erro de comunicação o Driver permanece em <i>loop</i> , recebendo dados até que ocorra um <i>time-out</i> . Isto limpa o canal de recepção e impede que ocorram problemas em futuras comunicações devido à recepção de bytes atrasados que ainda estejam trafegando no momento do erro, e que possam ser confundidos com uma resposta a um novo comando
<b>Reconnect after Timeout (Ethernet only)</b>	Com esta opção habilitada, após qualquer erro de <i>time-out</i> na recepção de <i>frames</i> do equipamento, o Driver promove a desconexão e a reconexão da camada física, limpando a conexão de possíveis <i>frames</i> atrasados que estejam em tráfego e que possam afetar futuras requisições. Esta opção deve sempre ser habilitada caso seja inevitável o uso de <b>Modbus RTU</b> em meio <b>Ethernet TCP/IP</b> em sistemas legados, uma vez que o modo <b>RTU</b> não possui controle de transação, portanto nem sempre é possível distinguir um <i>frame</i> de resposta correto de um outro <i>frame</i> atrasado resultante de leitura anterior, possivelmente de outro endereço, e que tenha falhado por <i>time-out</i> . Para novos projetos, recomenda-se fortemente que <b>NÃO</b> sejam utilizados os modos <b>Modbus RTU</b> ou <b>Modbus ASC</b> em meio <b>Ethernet TCP/IP</b> . Note que é preciso manter habilitada a opção <b>Retry failed connection every</b> da aba <b>Setup</b> do <b>IOMKit</b> para que o Driver se reconecte após o <i>time-out</i> . Caso contrário, o <i>time-out</i> apenas gera uma desconexão e cabe à aplicação o gerenciamento desta nova conexão

**NOTA**

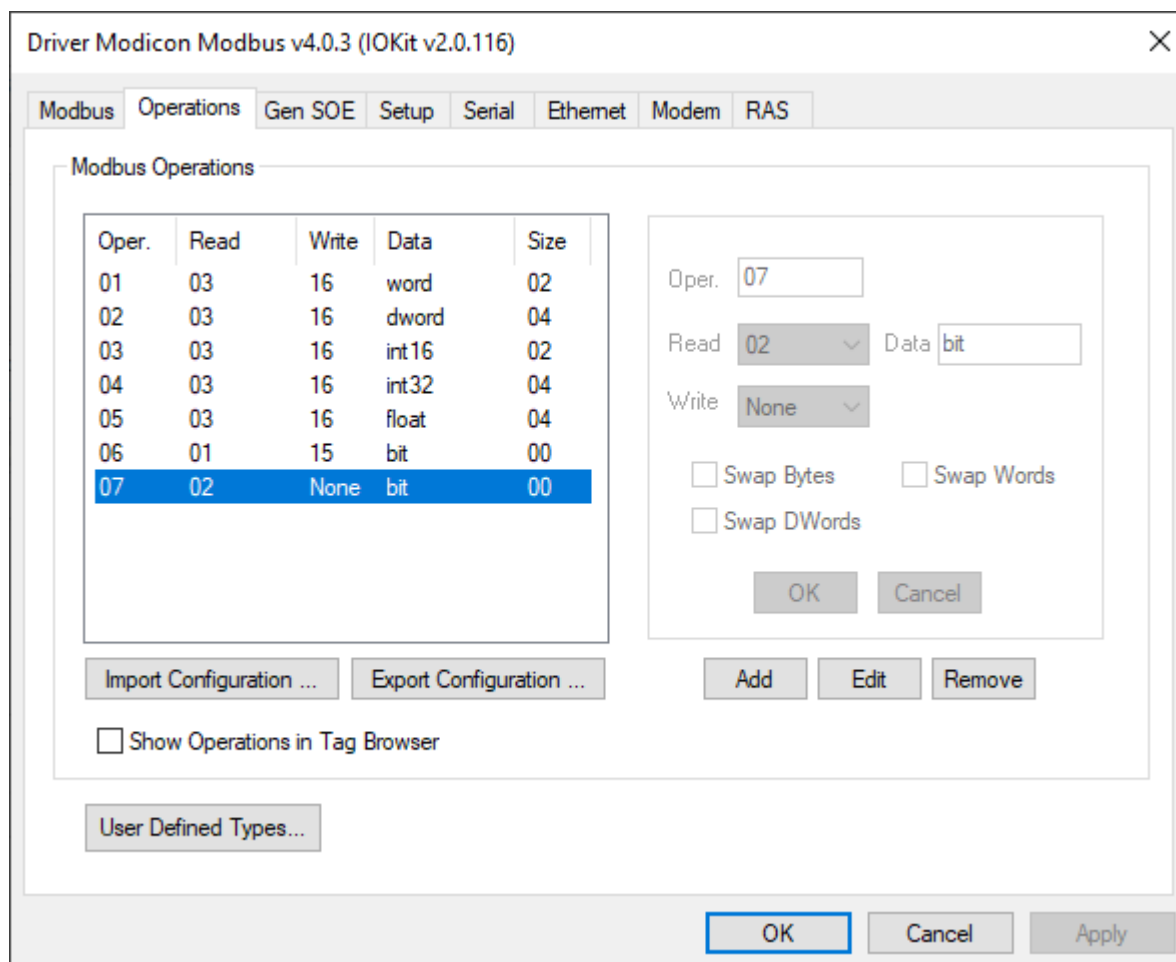
A antiga opção **Swap Address Delay** foi removida da janela de configuração na versão 2.08. O Driver ainda mantém suporte a esta opção em aplicações pré-existentes e permite habilitá-la por scripts (veja o tópico **Configuração em Modo Offline**). Para aplicações novas, recomenda-se utilizar a opção **Inter-frame Delay** da aba **Serial** do **IOMit**, que substitui esta antiga opção com vantagens.



## Aba Operations

Este tópico descreve a configuração da aba **Operations** da janela de configuração do Driver, onde são definidas as operações utilizadas nos Tags de Comunicação, mostrada na figura a seguir.

A configuração de operações não é mais usada na **configuração de Tags por Strings**, sendo usada apenas na antiga **configuração numérica** (parâmetros *N/B*) do **Elipse SCADA**.



Aba Operations na janela de configurações do Driver

## Operações

Para o correto funcionamento deste Driver, é necessário definir quais as funções Modbus de leitura ou escrita são utilizadas para cada Tag de Comunicação. Para isto, caso a configuração dos Tags seja realizada através dos antigos **parâmetros numéricos N/B** do **Elipse SCADA**, deve-se selecionar a aba **Operations** na janela de configurações.

Para este Driver, são chamadas **Operações** as configurações que definem como cada Tag de Comunicação executa a leitura e a escrita de dados no equipamento.

**Uma operação nada mais é do que a definição de um par de funções do protocolo, uma para escrita e outra para a leitura, e a especificação de conversões adicionais no formato dos dados que podem ser associados aos Tags da aplicação.** Em outras palavras, no Driver Modbus os parâmetros numéricos *N* ou *B* dos Tags de Comunicação não referenciam diretamente as funções do protocolo, mas sim operações pré-configuradas, que por sua vez não só informam as funções (**nativas** do protocolo ou mesmo **especiais**) a serem usadas na comunicação, como também a forma como os dados nativos do protocolo devem ser interpretados.

A configuração dos parâmetros dos Tags de Comunicação é descrita mais adiante no tópico **Configurando um Tag de Comunicação**. A seguir é descrita a configuração das operações, que mais tarde devem ser associadas a cada Tag de Comunicação.

**NOTA**

As operações funcionam apenas como modelos, ou *templates*, para a configuração de **Tags de Comunicação**, sendo possível, e em geral necessário, atribuir uma mesma operação a diversos Tags, que têm em comum o mesmo valor em seus parâmetros *N2/B2*.

**Funções**

O protocolo Modbus define funções de leitura e escrita, as quais podem acessar espaços de endereçamento distintos no equipamento, e com tipos de dados específicos. As funções **03** e **16** por exemplo, as mais usadas do protocolo, são responsáveis respectivamente pela leitura e escrita de *Holding Registers*, que nada mais são do que valores inteiros sem sinal de 16 bits (**Words**).

As funções do protocolo Modbus padrão fornecem dados apenas em formatos básicos de **Bit** e **Word** de 16 bits. Não existem formatos de dados adicionais na especificação do protocolo.

A lista das funções Modbus suportadas pelo Driver, e que podem ser atribuídas às operações configuradas, pode ser conferida no tópico **Funções Suportadas**.

Além das funções do protocolo, o Driver também contém algumas **Funções Especiais** que não fazem parte do protocolo padrão, de formato proprietário e utilizadas em geral para a leitura de eventos (SOE).

**Formatação de Dados**

Além de permitir a associação das funções (do protocolo ou especiais) a Tags específicos, as operações também permitem a definição de formatação adicional a ser aplicada aos dados, possibilitando o suporte a tipos de dados adicionais, não especificados pelo protocolo, como por exemplo valores de ponto flutuante de 32 bits (**Float**) e 64 bits (**Double**). Os tipos de dados suportados são descritos no tópico **Tipos de Dados Suportados**.

É importante observar que, quando tipos de dados de 32 e 64 bits são definidos nas operações, é necessário definir funções do protocolo que trabalhem com registros de 16 bits. Desta forma, a leitura de dados com mais de 16 bits resulta na leitura de vários registros Modbus de 16 bits do equipamento, ou seja, para a leitura de um Tag associado a uma operação que defina o tipo de dado **Float** de 32 bits, o Driver precisa ler dois registros consecutivos de 16 bits do equipamento, concatená-los e realizar a conversão para o formato **Float**.

Também é possível definir tipos de dados de oito bits (**Byte**, **Int8** ou **Char**) nas operações. Note que, uma vez que as funções do protocolo não permitem a leitura e escrita de bytes isolados, para cada dois Elementos de Bloco de tipos de dados de oito bits, o Driver é obrigado a acessar um registro distinto de 16 bits no equipamento. Por este motivo, o Driver não permite a escrita de tipos de dados de oito bits em Tags, em Elementos isolados de Bloco ou em Blocos de tamanho ímpar ou unitário. A escrita de tipos de dados de oito bits deve ser realizada sempre em Blocos de tamanho par.

**Tipos de Dados Definidos pelo Usuário**

Além dos tipos de dados pré-definidos (tipos de dados nativos ou *built-in*) descritos no tópico **Tipos de Dados Suportados**, este Driver permite também tipos de dados definidos pelo usuário. Estes tipos de dados devem ser declarados em janela específica, clicando-se em **User Defined Types** na parte inferior da aba **Operations**. Tais tipos de dados consistem em estruturas criadas a partir dos tipos de dados pré-definidos. Para mais informações sobre os tipos de dados definidos pelo usuário, consulte o tópico **Tipos de Dados Definidos pelo Usuário**.

**Byte Order**

Além das funções de leitura e escrita do protocolo e do tipo de dado utilizado, cada operação permite também atribuir manipulações adicionais aos bytes, relacionadas ao chamado *byte order*, ou seja, a ordem dos bytes dentro de cada valor. São as chamadas opções de *swap* (*Swap Bytes*, *Swap Words* e *Swap DWords*). Tais opções somente necessitam ser habilitadas no caso de equipamentos que não respeitem a ordem de bytes padrão do protocolo.

O protocolo Modbus define que seus valores de 16 bits utilizam sempre o *byte order* chamado de *big endian*, também conhecido como *Motorola*, por ser utilizado por este fabricante. O padrão *big endian* define sempre a ordem dos bytes de tal forma que o byte mais significativo de cada valor venha sempre antes. Desta forma, por exemplo, na leitura do valor hexadecimal 1234h, o equipamento envia primeiro o byte mais significativo 12h e logo a seguir o menos significativo, 34h.

No caso de equipamentos que não implementem o *byte order* padrão do protocolo e que utilizem o chamado *little endian* ou *Intel*, os dados são enviados com os bytes menos significativos antes. É preciso então habilitar as opções de *swap* para inverter a ordem dos bytes.

Há ainda equipamentos que usam *byte orders* diferentes para tipos de dados de 32 e 16 bits. No caso, por exemplo, de equipamentos que usem o *byte order* padrão do Modbus (*big endian*) para tipos de dados de 16 bits, porém forneçam dados de 32 bits com o **Word** menos significativo vindo primeiro (*little endian*), é necessário habilitar apenas a opção **Swap Words**, deixando desmarcada a opção **Swap Bytes**. Em suma, pode-se ter basicamente três situações:

- Caso o equipamento forneça dados usando o *byte order* padrão do protocolo Modbus (*Motorola* ou *big endian*), com os bytes mais significativos vindo antes, deve-se deixar as opções de *swap* todas desabilitadas. Esta é a situação mais comum
- Caso o equipamento use outro padrão de *byte order*, com os bytes menos significativos vindo antes (*little endian*), é necessário habilitar-se todas as opções de *swap* referentes ao tipo de dados usado, ou seja, para tipos de dados de 16 bits, habilite a opção **Swap Bytes**. Para tipos de dados de 32 bits, habilite as opções **Swap Bytes** e **Swap Words**. Para tipos de dados de 64 bits, as três opções de *swap* devem ser habilitadas
- No caso menos comum de equipamentos que usem *byte orders* diferentes para tamanhos de dados diferentes, fornecendo por exemplo o byte mais significativo de cada **Word** primeiro, porém o **Word** menos significativo de cada **DWord** primeiro, é preciso avaliar em qual caso cada opção de *swap* precisa ser habilitada, de forma a converter o valor retornado pelo equipamento para o formato *big endian* padrão do protocolo

## NOTA

As opções de *swap* citadas não têm efeito para tipos de dados **Bit** ou para tipos de dados com tamanho de oito bits (**Byte**, **Char** e **Int8**). A permuta ocorre dentro de cada tipo de dados, ou seja, a opção **Swap Words** não tem efeito para tipos de dados de 16 bits, assim como a opção **Swap DWords** não tem efeito para tipos de dados de 32 bits. Os tipos de dados **BCD** também não permitem *swaps*.

Para saber se o equipamento utiliza algum formato diferenciado de *byte order*, consulte a documentação do fabricante. Caso a informação não seja encontrada na documentação, o suporte técnico do fabricante deve ser contactado.

O tópico **Dúvidas Mais Frequentes** contém dicas de configurações de *byte order* para alguns equipamentos para os quais já se sabe ser necessário utilizar as opções de *swap*.

## Máscara de Bits

A opção **Use Bit Mask** é um recurso avançado, utilizada em casos mais específicos e raros em que o usuário deseja ler somente um bit do valor retornado pelo equipamento, mas não é possível usar o mapeamento de bits da aplicação.

Para a maioria dos usuários, os campos de mapeamento de bits da aplicação são a melhor alternativa para o acesso à máscaras de bits, não sendo preciso recorrer a este recurso do Driver.

Este recurso foi criado originalmente para permitir a leitura de bits de *Holding Registers* por bibliotecas especializadas do **E3**, em situações que impediam o uso do mapeamento de bits da aplicação.

Neste caso, o Driver lê normalmente o valor do equipamento e então o mascara, de forma a retornar ao campo **Valor** do Tag apenas o bit especificado (**0** ou **1**). A definição do número do bit a ser retornado é feita no parâmetro *N3/B3* do Tag de Comunicação.

A opção **Use Bit Mask** somente pode ser utilizada com tipos de dados inteiros de 16 bits ou mais (**Int16**, **Int32**, **Word** ou **DWord**). Além disto, operações que habilitam esta opção podem ser utilizadas apenas para a leitura. A função Modbus de escrita (**Write**) de operações que utilizam esta opção de máscara podem ser definidas como **None** (nenhuma).

## Operações Padrão do Driver

Por padrão, quando um novo Driver é adicionado à aplicação, este Driver já é criado com sete operações padrão, descritas na tabela a seguir.

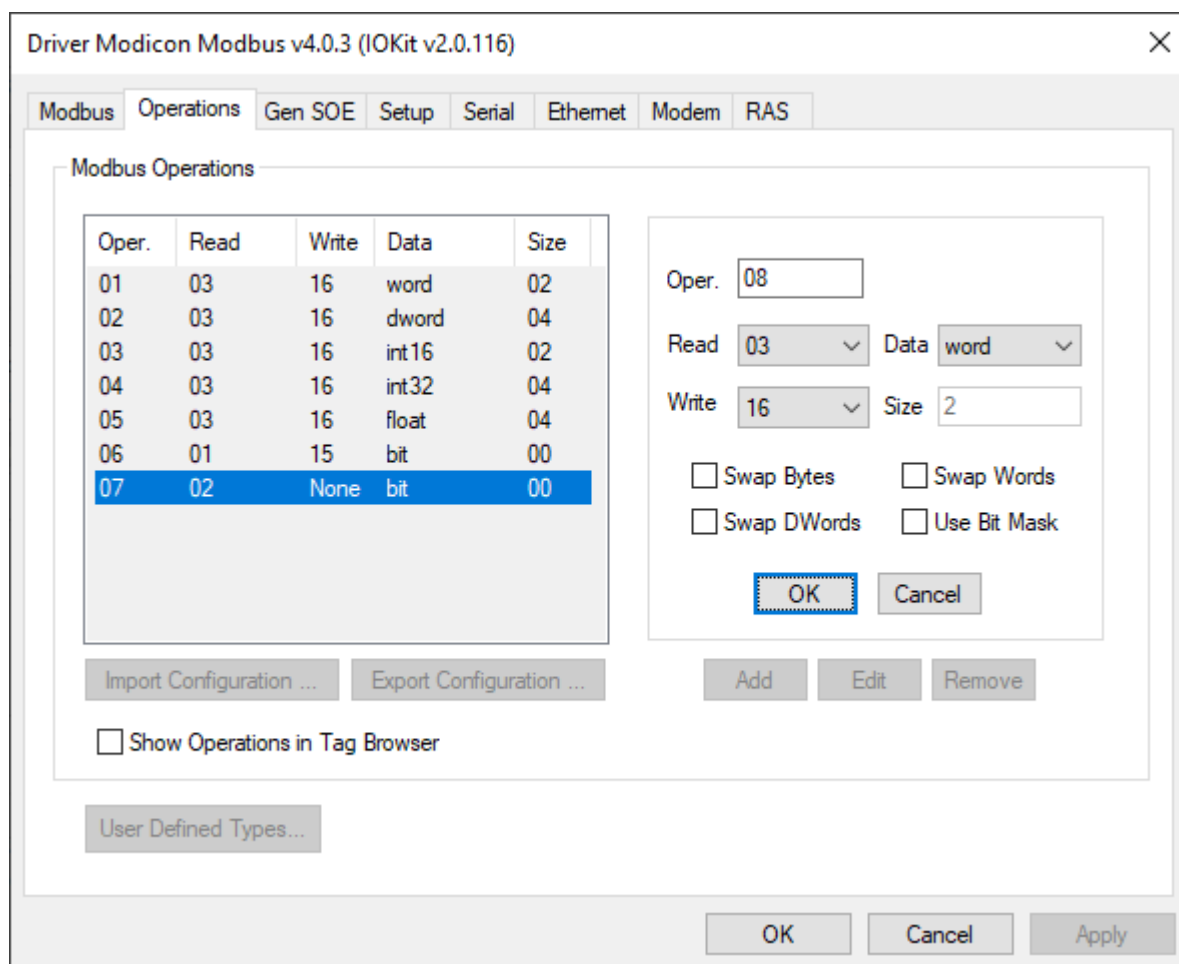
Operações padrão

OPERAÇÃO	FUNÇÃO DE LEITURA	FUNÇÃO DE ESCRITA	TIPO DE DADOS	FINALIDADE
<b>1</b>	3 - Read Holding Registers	16 - Write Multiple Registers	<b>Word</b>	Leitura e escrita de inteiros de 16 bits sem sinal
<b>2</b>	3 - Read Holding Registers	16 - Write Multiple Registers	<b>DWord</b>	Leitura e escrita de inteiros de 32 bits sem sinal
<b>3</b>	3 - Read Holding Registers	16 - Write Multiple Registers	<b>Int16</b>	Leitura e escrita de inteiros de 16 bits com sinal
<b>4</b>	3 - Read Holding Registers	16 - Write Multiple Registers	<b>Int32</b>	Leitura e escrita de inteiros de 32 bits com sinal
<b>5</b>	3 - Read Holding Registers	16 - Write Multiple Registers	<b>Float</b>	Leitura e escrita de valores de ponto flutuante de 32 bits
<b>6</b>	3 - Read Holding Registers	15 - Write Multiple Coils	<b>Bit</b>	Leitura e escrita de bits
<b>7</b>	2 - Read Discrete Inputs	None	<b>Bit</b>	Leitura de bits do bloco de dados de Entradas Discretas ( <i>Discrete Inputs</i> )

**Estas operações são as mais comumente usadas, sendo a operação 1 (um) a mais comum.** Para a maior parte dos equipamentos, selecione as operações necessárias entre as já fornecidas por padrão, não sendo necessário criar novas operações ou alterar a configuração das operações padrão.

## Definição de Novas Operações

Para adicionar uma nova operação no Driver, clique em **Add**.



#### Adicionando uma nova operação

Para configurar a nova operação, selecione um número para esta operação (este número é utilizado no parâmetro *N2/B2* dos Tags de Comunicação), qual função deseja utilizar para leitura e qual função deseja utilizar para escrita, além de informar o tipo de dados que é lido ou escrito pelo Driver. Note que, ao clicar em **Add**, o Driver já sugere um valor que ainda não esteja em uso para a nova operação.

Para mais informações sobre os tipos de dados suportados, veja o tópico **Tipos de Dados Suportados**. Os demais campos devem ser configurados conforme a necessidade. A tabela a seguir contém a descrição destes campos.

#### Opções de campos para operações

OPÇÃO	DESCRIÇÃO
<b>Size</b>	Deve ser informado o tamanho em bytes de cada elemento do tipo de dados selecionado. Este campo é preenchido automaticamente para tipos de dados com tamanho fixo, como <b>Byte</b> , <b>Word</b> e <b>Int16</b> , devendo ser preenchido para tipos de dados <b>String</b> e <b>BCD</b> . No caso de <b>Strings</b> , este tamanho define exatamente o número de bytes enviados ou recebidos para cada valor <b>String</b> , isto é, para cada Tag ou Elemento de Bloco. Se a <b>String</b> lida ou escrita tem um tamanho menor, o restante dos bytes é preenchido com zeros, de forma a completar o tamanho configurado. O tipo de dados <b>String</b> neste Driver não possui um limite máximo de tamanho definido, este limite é o <b>máximo permitido pelo protocolo</b> para a área de dados do <i>frame</i> de uma determinada função
<b>Swap Bytes</b>	Indica que o Driver deve inverter a ordem dos bytes, um a um, para obter o valor

OPÇÃO	DESCRIÇÃO
<b>Swap Words</b>	Indica que o Driver deve inverter a ordem dos bytes, dois a dois (em <b>Words</b> ), para obter o valor
<b>Swap DWords</b>	Indica que o Driver deve inverter a ordem dos bytes, quatro a quatro (em <b>DWords</b> ), para obter o valor
<b>Use Bit Mask</b>	Habilita o mascaramento de bits de registradores, através do parâmetro <i>N3/B3</i> . Esta opção afeta apenas a leitura e pode ser usada apenas com tipos de dados inteiros, com ou sem sinal, com pelo menos 16 bits de tamanho ( <b>Int16</b> , <b>Int32</b> , <b>Word</b> ou <b>DWord</b> ). Operações com esta opção habilitada não podem ser usadas para escrita. Para a maioria dos usuários, recomenda-se usar o mapeamento de bits da aplicação, deixando esta opção desmarcada (veja a <b>seção específica</b> )

As funções do protocolo que podem ser configuradas nos campos **Read** e **Write** das operações estão descritas no tópico **Funções Suportadas**. A tabela a seguir descreve cada uma das opções disponíveis.

#### Opções disponíveis na aba Operations

OPÇÃO	DESCRIÇÃO
<b>Import Configuration</b>	Esta opção permite importar configurações de operações de versões anteriores à 2.0 do Driver Modbus Master/Slave, que armazenavam estas configurações em um arquivo modbus.ini. Este Driver não utiliza mais arquivos INI para armazenar tais configurações, que agora são armazenadas no próprio arquivo da aplicação. Para maiores detalhes, consulte o tópico <b>Importação e Exportação de Operações</b>
<b>Export Configuration</b>	Esta opção executa a operação inversa da anterior, gerando um arquivo INI contendo as configurações de operações, no formato atual ou no mesmo formato das versões anteriores deste Driver. Desta forma, é possível guardar em um arquivo as configurações de operações de um determinado equipamento, que podem ser utilizadas em outras aplicações. Para maiores detalhes, consulte o tópico <b>Importação e Exportação de Operações</b>
<b>Show Operations in Tag Browser</b>	Se esta opção não estiver selecionada (padrão), são mostrados modelos de <b>Tags configurados por Strings</b> (campos <b>Dispositivo</b> e <b>Item</b> ) no Tag Browser. Se estiver selecionada, modelos de <b>Tags configurados numericamente</b> (parâmetros <i>N/B</i> ), representando as operações configuradas, são mostrados no Tag Browser. Quando novas instâncias do Driver são criadas, esta opção vem desmarcada por padrão. Em aplicações legadas, quando a versão do Driver é atualizada a partir de uma versão anterior à 3.1, a opção já vem selecionada, mantendo o comportamento das versões anteriores
<b>Add</b>	Adiciona uma nova operação à lista
<b>Edit</b>	Atualiza a operação selecionada na lista (equivale a clicar duas vezes no item)

OPÇÃO	DESCRIÇÃO
Remove	Remove a operação selecionada na lista

#### NOTA

As opções **Swap Bytes**, **Swap Words** e **Swap DWords**, conforme já explicado, foram acrescentadas para permitir compatibilidade com equipamentos que não seguem o padrão do protocolo Modbus na codificação dos dados (*byte order*). **Se estas opções permanecerem desabilitadas, o comportamento do Driver corresponde ao padrão do protocolo, sendo esta a opção recomendada para a maioria dos equipamentos.**

## Tipos de Dados Suportados

A tabela a seguir relaciona os tipos de dados nativos deste Driver que podem ser definidos na configuração dos Tags.

Conforme explicado nos tópicos **Configuração por Strings** e **Aba Operations**, o protocolo Modbus em si tem suporte apenas aos tipos de dados **Bit** e **Word** (16 bits) para as **funções mais comuns** implementadas neste Driver (a exceção atualmente é a **função 7**). Todos os demais tipos de dados deste Driver são convertidos para **Word** no nível do protocolo, para a leitura ou escrita no equipamento ou dispositivo escravo.

Vale lembrar que este Driver também suporta os **Tipos de Dados Definidos pelo Usuário**, definidos como estruturas com elementos compostos com os tipos de dados nativos da tabela a seguir.

Na tabela a seguir, os tipos de dados usam as mesmas denominações dos mnemônicos para o campo **tipos de dados**, quando os Tags são **configurados por Strings**. Na **configuração numérica**, as mesmas denominações são também utilizadas na coluna **Data** da janela de configuração do Driver, na **Aba Operations**. Em alguns casos, denominações alternativas frequentes são apresentadas entre parênteses.

#### Opções disponíveis para tipos de dados

TIPO	FAIXA	DESCRIÇÃO
Char	-128 a 127	Palavra de oito bits, caractere. A escrita deve ocorrer sempre em blocos de tamanho par ( <b>Words</b> )
Byte	0 a 255	Palavra de oito bits sem sinal. A escrita deve ocorrer sempre em blocos de tamanho par ( <b>Words</b> )
Int8	-128 a 127	Palavra de oito bits com sinal. A escrita deve ocorrer sempre em blocos de tamanho par ( <b>Words</b> )
Int16	-32768 a 32767	Inteiro de 16 bits com sinal
Int32	-2147483648 a 2147483647	Inteiro de 32 bits com sinal
Word (ou UInt)	0 a 65535	Inteiro de 16 bits sem sinal
DWord (ou UInt)	0 a 4294967295	Inteiro de 32 bits sem sinal ( <b>Double Word</b> )
Float	-3.4E38 a 3.4E38	Ponto flutuante de 32 bits ( <b>IEEE 754</b> ) (quatro bytes: EXP F2 F1 0)

TIPO	FAIXA	DESCRIÇÃO
<b>Float_GE</b>	-1.427E+45 a 1.427E+45	Ponto flutuante de 32 bits usado pela GE, não compatível com <b>IEEE 754</b> . É usado em equipamentos GE GEDE UPS, com expoente de oito bits $2^{[-128 \dots + 127]}$ e 24 bits de mantissa $[-2^{23} \dots + (2^{23} - 1)]$ . (quatro bytes: EXP F2 F1 F0). Para mais informações, consulte a documentação do equipamento
<b>Double (ou Real)</b>	-1.7E308 a 1.7E308	Ponto flutuante de 64 bits ( <b>IEEE 754</b> )
<b>String</b>	Não se aplica	Texto em formato <b>ANSI</b> , com número determinado de caracteres ASCII de oito bits ( <b>Chars</b> )
<b>BCD</b>	Ver descrição	Valor numérico BCD ( <i>Binary-Coded Decimal</i> ou <i>Decimal Codificado em Binário</i> ). Ao utilizar este tipo de dados, a aplicação deve fornecer um valor decimal positivo e inteiro, a ser enviado no formato <b>BCD</b> , respeitando o tamanho especificado. O campo <b>Size</b> , no caso do tipo de dados <b>BCD</b> , refere-se ao número de bytes a serem enviados para representar o valor. Uma vez que na codificação <b>BCD</b> cada algarismo é convertido em um <i>nibble</i> , tem-se que os valores permitidos devem possuir um número máximo de algarismos igual ao dobro do valor especificado no campo <b>Size</b> , ou seja, se for selecionado o valor dois para o campo <b>Size</b> , o máximo valor que pode ser enviado é 9999. Já se <b>Size</b> é igual a quatro, o valor máximo é 99999999. Os valores permitidos para o campo <b>Size</b> no caso de tipos de dados <b>BCD</b> são dois ( <b>Word</b> ) e quatro ( <b>Double Word</b> ). Para mais detalhes sobre a codificação <b>BCD</b> , consulte o tópico <b>Codificação BCD</b>
<b>GE_events</b>	Ver descrição	Tipo de dados utilizado na leitura do <i>buffer</i> de eventos (SOE) do CLP GE PAC RX7. Sua definição é permitida apenas em operações que utilizem a função especial de leitura <b>GE SOE</b> . Estes eventos são retornados como blocos de dois Elementos, com <i>timestamps</i> definidos pelo controlador. Para mais informações, veja o tópico <b>Leitura de Buffer de Eventos em controladores GE PAC RX7</b>



TIPO	FAIXA	DESCRIÇÃO
<b>Bit</b>	0 (zero) ou 1 (um)	Este tipo de dados é selecionado automaticamente quando uma função de acesso a bits é selecionada. As funções de acesso a bits são <b>01</b> , <b>02</b> , <b>05</b> e <b>15</b> . O campo <b>Size</b> não é usado para tipos de dados <b>Bit</b> . Quando este tipo de dados é usado, cada Tag ou Elemento de Tag Bloco passa a representar um bit
<b>SP_events</b>	Ver descrição	Tipo de dados utilizado na leitura de eventos (SOE) de relés Schneider Electric das séries SEPAM 20, 40 e 80. Sua definição só é permitida quando a operação utilizar como função de leitura a função especial <b>SP SOE</b> . Estes eventos são retornados como um Bloco de três Elementos, com <i>timestamp</i> fornecido pelo equipamento. Para mais informações, veja o tópico <b>Leitura de Eventos de Relés Schneider Electric SEPAM 20, 40 e 80</b>
<b>GenTime</b>	1/1/1970 00:00 a 31/12/2035 23:59:59.999 (ver nota a seguir)	Tipo de dados de data e hora composto por uma estrutura de oito bytes, criado originalmente para ser utilizado na leitura de eventos que usam o <b>algoritmo de SOE Genérico (GenSOE)</b> . Este tipo de dados pode ser utilizado com as demais funções do protocolo Modbus, além da <b>GenSOE</b> . Como este formato é lido internamente como uma estrutura de <b>Words</b> , a única função de <i>swap</i> válida para este tipo de dados é <b>Swap Bytes</b> . A representação deste tipo de dados na memória do CLP é descrita no tópico <b>Tipo de Dados GenTime</b> . Para mais informações sobre este tipo de dados, consulte o tópico <b>Algoritmo de Leitura de SOE Genérico da Elipse Software</b>
<b>Sp_time</b>	1/1/1970 00:00 a 31/12/2035 23:59:59.999 (ver nota a seguir)	Tipo de dados de data e hora composto por uma estrutura de oito bytes, utilizado por relés Schneider Electric das séries SEPAM 20, 40 e 80, geralmente para representar um <i>timestamp</i> . Para mais informações, consulte a documentação do equipamento
<b>UTC64d</b>	1/1/1970 00:00 a 31/12/2035 23:59:59.999 (ver nota a seguir)	Tipo de dados de data e hora representado em formato <b>Double (IEEE 754 64 bits)</b> , com os segundos desde 1/1/1970 00:00

TIPO	FAIXA	DESCRIÇÃO
<b>UTC32</b>	1/1/1970 00:00 a 31/12/2035 23:59:59.999 (ver nota a seguir)	Tipo de dados de data e hora representado em formato inteiro sem sinal de 32 bits ( <b>DWord</b> ou <b>UInt</b> ), com os segundos desde 1/1/1970 00:00. Neste formato não são representados os milissegundos, sendo considerados sempre 0 (zero)
<b>Int16_sm</b>	-32767 a 32767	Inteiro de 16 bits com sinal (sinal de magnitude)
<b>Int32_sm</b>	-2147483647 a 2147483647	Inteiro de 32 bits com sinal (sinal de magnitude)
<b>Int52 e UInt52</b>	-9007199254740992 a 9007199254740992	Inteiros de 52 bits convertidos para <b>Double</b> (ponto flutuante)

#### NOTAS

- Embora a representação em si dos tipos de dados de data e hora da tabela anterior possa representar datas superiores a 31/12/2035, este limite é mostrado na tabela pelo fato de os aplicativos da **Eclipse Software** não possuírem suporte, atualmente, para faixas de valores superiores a este limite em estampas de tempo.
- Os tipos **Int52** e **UInt52** manipulam dados de forma equivalente aos Inteiros de 64 bits, ou seja, manipulam 4 (quatro) registros Modbus de 8 (oito) bytes por valor, limitando-se à faixa de valores representado pelos Inteiros de 52 bits, em função da conversão para o tipo de dados **Double**. Este Driver não suporta dados inteiros de 64 bits nativamente.

## Tipo de Dados GenTime

**GenTime** é um tipo de dados de data e hora definido e adicionado originalmente ao Driver para uso com o **Algoritmo de Leitura de SOE Genérico da Elipse Software**. Trata-se entretanto de um tipo de dados genérico, que pode ser usado com praticamente qualquer CLP, de forma simples.

Na aplicação do supervisório, ou seja, nos valores dos Tags e Elementos de Bloco de Tags Bloco, bem como no campo **Timestamp** dos Tags, este tipo de dados, como aliás todos os demais tipos de dados de data e hora do Driver, é representado por um tipo de dados de data e hora nativo da aplicação. Para mais informações sobre os demais tipos de dados de data e hora suportados pelo Driver, consulte o tópico **Tipos de Dados Suportados**. Para mais informações sobre os tipos de dados de data e hora da aplicação, consulte o respectivo manual do usuário (existem algumas diferenças do **Elipse SCADA** para o VBScript utilizado no **E3** e **Elipse Power**).

No CLP ou dispositivo escravo, este tipo de dados é representado por uma estrutura composta por quatro registradores de 16 bits (oito bytes), conforme mostrado na tabela a seguir.

**Estrutura dos registradores**

OFFSET	CONTEÚDO	MAPA DE BITS (16 BITS)	FAIXA (DECIMAL)
<b>0</b>	Ano	AAAAAAAA AAAAAAAAAA	Entre 0 e 65535
<b>1</b>	Dia e Mês	DDDDDDDD MMMMMMMM	Entre 0 e 65535
<b>2</b>	Hora e Minuto	HHHHHHHH MMMMMMMM	Entre 0 e 65535
<b>3</b>	Segundo e Milissegundo	SSSSSSMM MMMMMMMM	Entre 0 e 65535

O endereço base (*offset* 0), a ser atribuído no parâmetro *N4/B4* do Tag que acessa o dado, contém o ano. O registro seguinte (*offset* 1) tem o dia como o byte mais significativo e o mês como o byte menos significativo. Já no *offset* 2 tem-se a hora representada no byte mais significativo e os minutos no byte menos significativo. O quarto registro tem os quatro bits mais significativos do **Word** representando os segundos, e os bits restantes (os dois menos significativos do byte mais significativo e o byte menos significativo do inteiro) representando os milissegundos.

Note que cada Tag que referencie este tipo de dados força o Driver a ler um bloco de quatro registros Modbus no equipamento para representar o valor de cada Tag ou Elemento de Bloco para retornar um valor válido.

As vantagens deste tipo de dados são sua simplicidade (é fácil de gerar no *ladder* do CLP), sua precisão de milissegundos e sua relativa compactação, não necessitando de suporte nativo no CLP ou dispositivo escravo.

### NOTA

Embora o tipo de dados **GenTime** em si tenha um tamanho de oito bytes (quatro **Words**), a única opção de *swap* que tem efeito sobre ele é **Swap Bytes**. Isto porque, conforme visto neste tópico, este tipo de dados é estruturado na memória do CLP como tendo quatro **Words**, não sendo ele mesmo um tipo de dados nativo do equipamento, e sim do Driver. Mais informações sobre as opções de *swap* (*byte order*) podem ser encontradas no tópico **Aba Operations**.

## Tipos de Dados Definidos pelo Usuário

Os tipos de dados definidos pelo usuário, ou estruturas, após configurados na janela de configuração **User Defined Types**, podem ser usados nas operações do Driver da mesma forma que os tipos de dados pré-definidos.

Estes tipos de dados são na verdade estruturas cujos elementos podem ter **tipos de dados nativos** diferentes, ou seja, um tipo de dados definido pelo usuário nada mais é do que uma estrutura definida a partir dos tipos de dados pré-definidos pelo Driver (tipos de dados nativos ou *built-in*), permitindo ao usuário configurar Tags Bloco onde cada Elemento pode ter um tipo de dados nativo diferente.

O usuário pode utilizar praticamente todos os tipos de dados pré-definidos pelo Driver em suas estruturas. Somente não são permitidos os tipos de dados **Bit**, os tipos de dados de oito bits, os tipos de dados de tamanho variável, como **String** e **BCD**, e tipos de dados de evento associados à funções específicas de SOE.

Uma vez tendo definido um tipo de dados, o usuário pode associá-lo a qualquer Tag, desde que ela utilize funções Modbus que suportem **Words**, ou seja, não é permitida a associação de um tipo de dados definido pelo usuário a uma operação que defina como função de leitura (**Read**) a função **01**, por exemplo, uma vez que esta lê apenas bits.

Além da definição dos elementos da estrutura, cujos valores são retornados em Elementos de Bloco, o usuário pode também definir o tipo de *timestamp* do Tag, bem como o endereço padrão para a estrutura, endereço que é usado para o parâmetro *B4* dos Tags disponíveis via Tag Browser do **E3**.

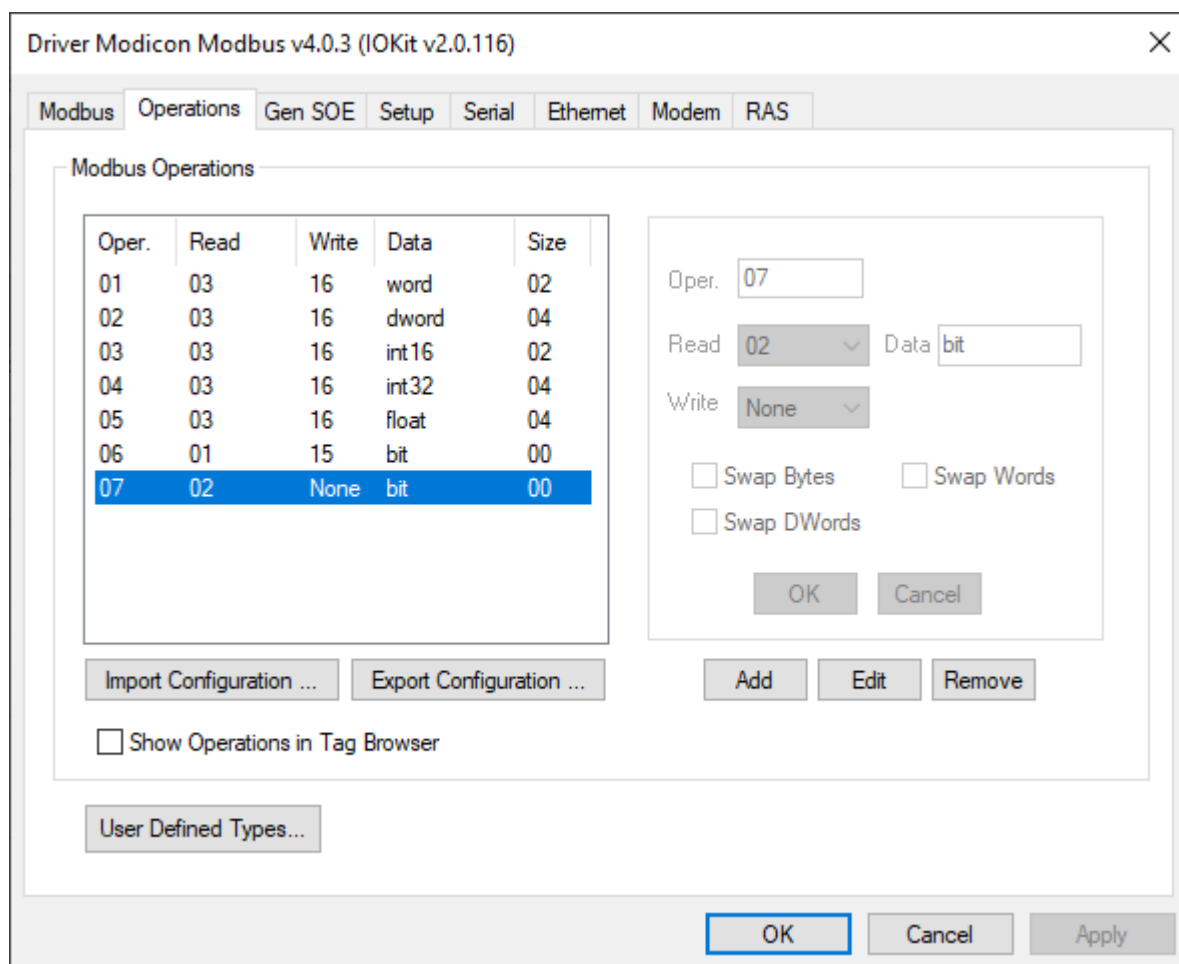
## Aplicações

Os tipos de dados definidos pelo usuário foram originalmente implementados para uso em conjunto com a **Rotina de SOE Genérico do Driver Modbus (Gen SOE)**, uma vez que esta rotina executa a leitura de tabelas de estruturas de dados.

Além de poder ser utilizado com a rotina genérica de SOE, este recurso pode também ser usado para agrupar tipos de dados diferentes em um mesmo Tag Bloco, otimizando a comunicação em aplicações que não contam com o recurso de Superblocos, caso do **Elipse SCADA**, ou caso o equipamento em uso por algum motivo não permita o uso de Superblocos (veja o tópico **Leitura por Superblocos**).

## Configuração de Tipos de Dados Definidos pelo Usuário

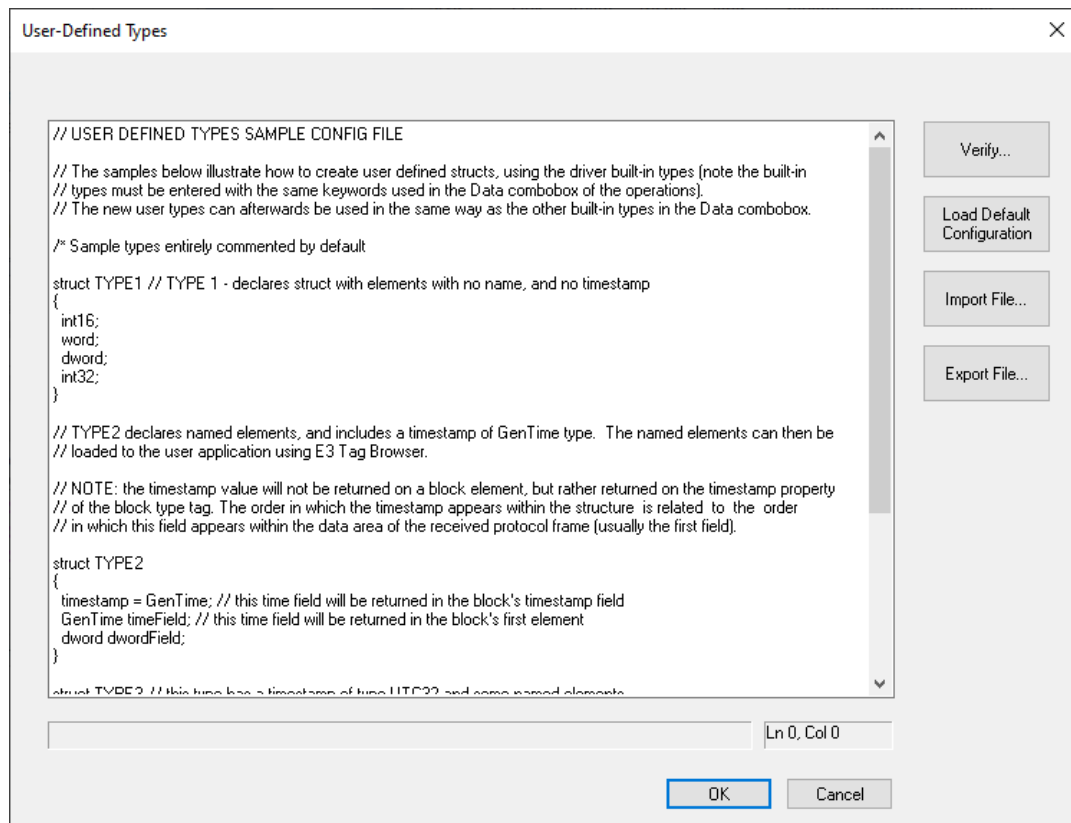
A configuração de tipos de dados definidos pelo usuário é realizada em janela específica, clicando em **User Defined Types** na **Aba Operations** da janela de configurações do Driver, conforme a figura a seguir.



**Aba Operations da janela de configurações do Driver**

A janela de configuração dos tipos de dados definidos pelo usuário permite a edição do arquivo de configuração das estruturas. Ao abrir a janela pela primeira vez, é mostrado o arquivo de configuração padrão (com comentários), e que define três tipos de dados de exemplo, que aparecem comentados por comentários de múltiplas linhas ("/\*" e "\*/"), conforme explicado a seguir.

A figura a seguir mostra a janela de configuração de tipos de dados definidos pelo usuário, com um pequeno arquivo definindo os três tipos de dados de exemplo.



### Configuração de tipos de dados definidos pelo usuário

Note que os comentários de linha iniciam sempre com "//", identificando tudo o que vier à direita, na mesma linha, como comentário, seguindo o padrão dos comentários de linha da linguagem de programação C++, também usado por outras linguagens como Java e C#.

Também são suportados comentários de múltiplas linhas, seguindo novamente a mesma sintaxe do C++, iniciando por "/\*" e terminando com "\*/". Note que o arquivo exemplo que vem com o Driver já aplica este formato de comentário a seus tipos de dados de exemplo, deixando-os comentados por padrão. Remova as linhas indicadas por "/\* Sample types entirely commented by default" e "\*/" (sem as aspas) para que os três tipos de dados de exemplos estejam prontos para o uso.

À medida que o texto do arquivo de configuração é alterado, a barra de status mostra o resultado da análise sintática do arquivo, em tempo real. Esta barra de status mostra a mensagem "Status: OK!" se não forem detectados erros no arquivo.

A cada momento, a linha e a coluna da posição do cursor na caixa de edição são sempre mostradas no lado direito da barra de status. Os erros mostrados na barra de status sempre referenciam ao número da linha e da coluna onde ele foi detectado.

A verificação pode ser feita na íntegra também clicando-se em **Verify** e, em caso de erro, o cursor já é posicionado automaticamente na linha do erro.

A definição de cada tipo tem a seguinte sintaxe (os elementos entre colchetes são opcionais):

```
struct <Nome do Tipo>
{
    [timestamp = <tipo data e hora>;]
    [DefaultAddress = <endereço>;]
    <tipo> [nome do elemento 1];
    <tipo> [nome do elemento 2];
    <tipo> [nome do elemento 3];
    [...]
    <tipo> [nome do elemento n];
}
```

Onde:

- **struct**: Palavra-chave, em letras minúsculas, que inicia a definição do tipo de dados definido pelo usuário
- **<Nome do Tipo>**: Nome pelo qual o novo tipo de dados é identificado pelo Driver. Este é o nome mostrado na caixa de seleção **Data**, na configuração das operações. Deve ter no máximo seis caracteres
- **timestamp**: Campo opcional que indica que a estrutura tem um *timestamp* definido pelo dispositivo, que deve ser retornado no campo **Timestamp** do Tag. Cada estrutura pode ter no máximo um *timestamp*. A ordem em que aparece na estrutura influencia a posição em que o campo é lido no *frame* retornado pelo equipamento (note que nos Tags este valor é retornado somente no campo **Timestamp**). Podem ser definidos quaisquer tipos de dados de data e hora suportados pelo Driver. Na versão atual, o Driver suporta os tipos de dados de data e hora **GenTime**, **Sp\_time**, **UTC64d** e **UTC32**. Para mais informações sobre tipos de dados, consulte o tópico **Tipos de Dados Suportados**
- **DefaultAddress**: Campo opcional que especifica um valor de endereço padrão, usado para preencher o parâmetro **B4** dos Tags no Tag Browser que referenciem as operações contendo a estrutura definida. Os valores de endereço podem ser fornecidos em decimal ou em hexadecimal. Para usar este último, é preciso preceder o número com o prefixo "0x" (por exemplo, usar "0x10" para codificar o valor decimal 16 em hexadecimal)
- **<tipo data e hora>**: Tipos de dados de data e hora pré-definidos pelo Driver, que podem ser utilizados como *timestamp* pelo dispositivo escravo. Na atual versão do Driver, são aceitos os tipos de dados nativos **GenTime**, **Sp\_time**, **UTC32** e **UTC64d**
- **<tipo>**: Tipo de dados do elemento. Deve ser definido como um dos tipos de dados pré-definidos pelo Driver, e escrito da mesma forma que aparece na caixa de seleção **Data**, no *frame* de configuração dos parâmetros das operações, considerando letras maiúsculas e minúsculas. Não são permitidos tipos de dados **Bit**, tipos de dados de oito bits e nem tipos de dados de tamanho variável, como **BCD** e **String**
- **[nome do elemento]**: Parâmetro opcional que define o nome de cada Elemento do Bloco. Se definido, determina o nome dos Elementos de Bloco nos Tags presentes no Tag Browser do **E3**. Se não for definido na declaração da estrutura, o Driver atribui nomes padrão aos Elementos no Tag Browser, com a palavra-chave "Element" seguida do valor do índice do Elemento dentro do Bloco ("Element1", "Element2", etc.)

## Importação e Exportação

As opções **Import File** e **Export File** permitem importar e exportar o arquivo de configuração de tipos de dados definidos pelo usuário para arquivos texto em disco. Podem ser utilizados para a realização de cópias de segurança do arquivo, ou para compartilhá-lo entre vários Drivers. O arquivo é sempre gravado e lido no formato ANSI padrão do Windows (Charset **Windows-1252**). Futuras versões do Driver podem suportar outros formatos.

Além de copiar o arquivo para o disco, pode-se também usar as teclas de atalho CTRL + A (**Selecionar Tudo**), CTRL + C (**Copiar**) e CTRL + V (**Colar**) para copiar e colar o conteúdo do arquivo em outro Editor de Texto.

A opção **Load Default Configuration** carrega novamente no editor o arquivo padrão de configuração, o mesmo que já vem carregado no editor quando a janela de configuração é aberta pela primeira vez.

### NOTA

Ao clicar em **Cancel**, todas as alterações realizadas no arquivo são descartadas pelo Driver. Clicando-se em **OK**, o arquivo é armazenado na aplicação. Esta operação realiza a verificação completa do arquivo e, se for identificado algum erro, este é mostrado e a janela não é fechada. Se for preciso salvar alterações com erros ainda pendentes, exporte o arquivo ou copie-o e cole-o em outro Editor de Textos.

## Utilizando Tipos de Dados Definidos pelo Usuário na Configuração por Strings

Pode-se fornecer nomes de tipos de dados definidos pelo usuário como mnemônicos do campo **Tipo** do campo **Item**, como ocorre com os tipos de dados nativos do Driver, desde que o nome tenha sido previamente declarado, como explicado anteriormente neste tópico.

#### IMPORTANTE

Como no **E3** o campo **Item** não diferencia entre minúsculas e maiúsculas, para usar tipos de dados definidos pelo usuário neste campo é necessário que os nomes dos tipos de dados definidos não difiram apenas pela caixa alta ou baixa, ou seja, não se deve definir, por exemplo, um tipo de dados com o nome "tipo1" e outro com o nome "TIPO1". Caso isto ocorra, não é possível usar tipos de dados definidos pelo usuário no campo **Item** até que os nomes sejam corrigidos.

Para mais informações sobre a configuração de Tags usando **Strings**, consulte o tópico **Configuração por Strings**. Exemplo:

- Leitura ou escrita de *Holding Registers* (funções **03** e **06**) de endereço 100 do equipamento com *Id* 5, interpretado como um tipo de dados definido pelo usuário chamado "mytype", com *Slave Id* no campo **Item**:
  - **Dispositivo**: "" (empty **String**)
  - **Item**: "5:shr100.mytype"

#### NOTA

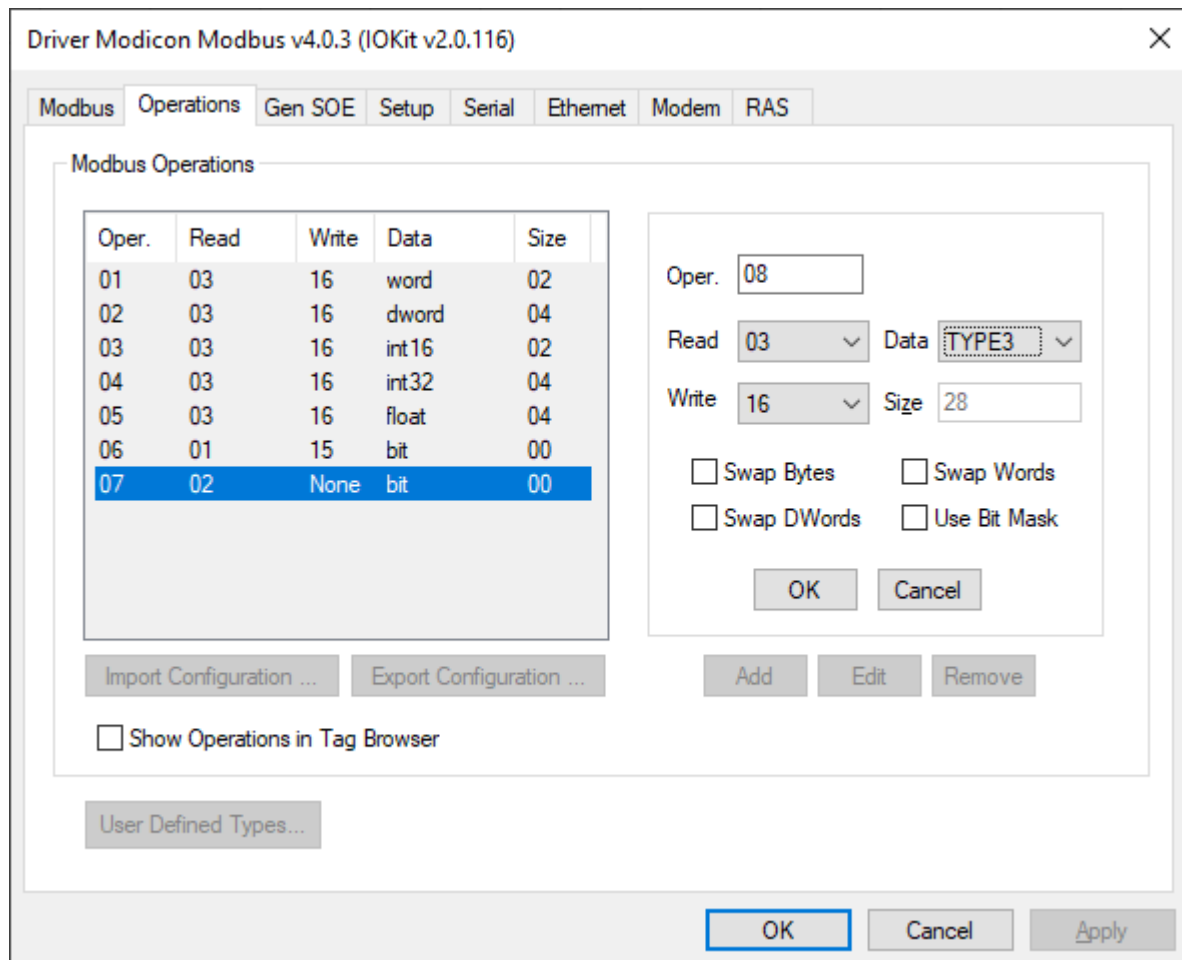
As opções de permuta (ordenamento de bytes) para tipos de dados definidos pelo usuário têm efeito apenas nos elementos da estrutura definida e não na estrutura inteira, ou seja, se a opção **Swap Words** está habilitada, todos os elementos com mais de 16 bits têm seus **Words** permutados. Os elementos de 16 bits, entretanto, não são alterados.

### Utilizando Tipos Definidos pelo Usuário na Configuração Numérica

Após a definição dos novos tipos de dados no arquivo de configuração na janela **User-Defined Types**, estes tipos de dados estão disponíveis para uso nas operações do Driver. Lembre-se que somente as operações que utilizam funções Modbus para acesso a registradores de 16 bits, como por exemplo as funções **03**, **04**, **06** e **16**, permitem tipos de dados definidos pelo usuário.

A figura a seguir mostra a configuração de uma nova operação que utiliza o tipo de dados definido pelo usuário (estrutura) de nome **TYPE3**, mostrado no exemplo acima, após o usuário clicar em **Add**.



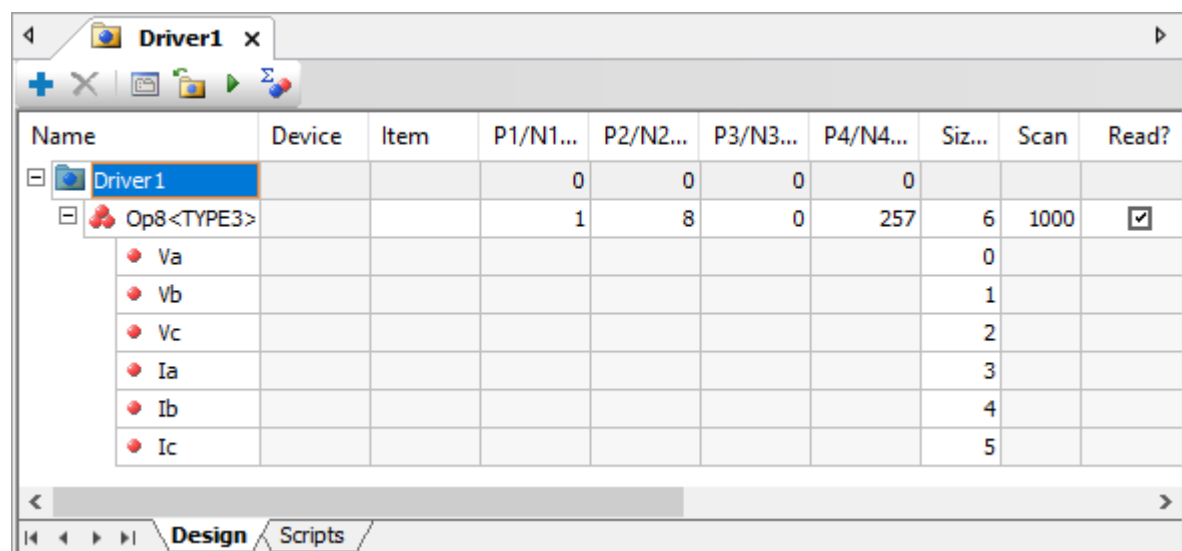


Adicionar tipo de dados definido pelo usuário

#### NOTA

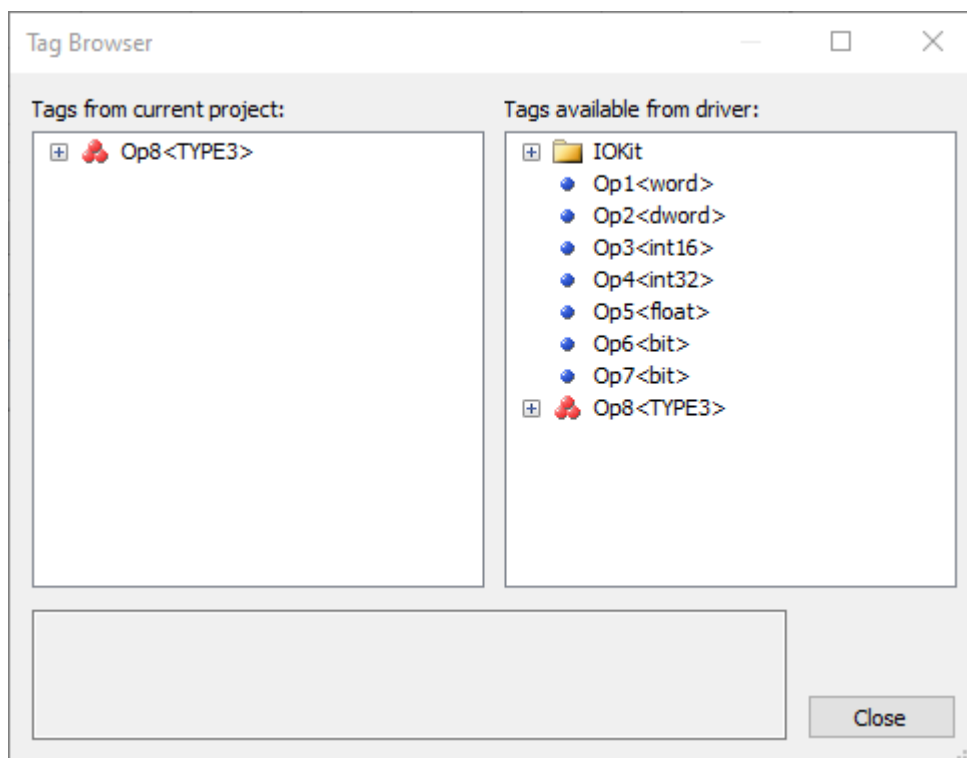
As opções de permuta para tipos de dados definidos pelo usuário têm efeito apenas nos elementos da estrutura definida e não na estrutura inteira, ou seja, se a opção **Swap Words** está habilitada, todos os elementos com mais de 16 bits têm seus **Words** permutados. Os elementos de 16 bits, entretanto, não são alterados.

Após a definição da nova operação, usando o novo tipo **TYPE3**, defina um Tag Bloco com o mesmo tipo de dados e com tamanho igual ao número de elementos da estrutura, conforme mostrado na figura a seguir.




Declaração de Tags usando estruturas no E3 ou Elipse Power

Se foi definido o nome para cada elemento da estrutura, é possível utilizar o Tag Browser do **E3** para incluir um Tag Bloco referente ao tipo de dados desejado na aplicação, sem precisar digitar novamente. Para utilizar este recurso, é necessário selecionar a opção **Show Operations in Tag Browser** na aba **Operations**. A figura a seguir ilustra o procedimento.



Uso do Tag Browser para definir Tags usando estruturas

Como a figura sugere, clique em  na aba **Design** do Driver para abrir o Tag Browser e arraste o tipo de dados desejado da lista **Tags disponibilizados pelo Driver** (*Tags available from driver*) para a lista **Tags do projeto corrente** (*Tags from current project*).

## Leitura Reportada a Eventos

Os tipos de dados definidos pelo usuário ou estruturas são comumente utilizados para a definição de eventos na memória do CLP, podendo ser usados com o **Algoritmo de Leitura de SOE Genérico da Elipse Software**. Se entretanto for necessário ler eventos organizados na memória do CLP, como uma sequência de estruturas, em uma operação que utilize apenas **função pública de leitura do protocolo**, ou seja, sem o emprego de funções especiais com algoritmo de SOE, tal procedimento pode ser realizado de duas formas:

- **Leitura em Bloco:** Crie um Bloco com um número de Elementos que seja múltiplo do número de elementos das estruturas de dados do usuário. Por exemplo, um tipo de dados definido pelo usuário ou estrutura com dois elementos que represente eventos acumulados em um arranjo na memória do CLP. Caso se deseje ler em bloco cinco eventos, é necessário definir um Tag Bloco contendo 10 Elementos. Assim, uma única leitura neste Tag traz todos os eventos de uma só vez
- **Leitura Reportada a Evento:** Usa uma sequência de eventos **OnRead** do Tag para ler o bloco de dados. Com isto, considerando o exemplo do item anterior, ao invés de criar um Tag com 10 Elementos, o usuário precisa criar apenas um único Tag Bloco com dois Elementos, configurando o parâmetro *B3* com o valor "5". Desta forma, ao realizar a leitura do Tag, o **E3** chama cinco vezes o evento **OnRead** do Tag, e em cada chamada os Elementos e propriedades do Tag Bloco contêm dados relativos a um evento específico. O uso mais comum dos Tags reportados a evento é o armazenamento dos eventos lidos diretamente na base de dados de histórico. Isto é facilmente executado através do método **WriteRecord** do objeto Histórico previamente associado ao Tag, dentro do evento **OnRead** do Tag reportado a evento. Para mais informações, consulte o tópico sobre Tags Reportados a Eventos no *Manual do Usuário do E3*

Em outras palavras, todo Tag de Comunicação que use estruturas e que utilize uma **função pública de leitura do protocolo** (este recurso não funciona para **funções especiais de SOE**), torna-se um Tag Reportado a Eventos se o seu parâmetro *B3* for configurado com um valor não nulo.

No caso de funções especiais de SOE, como a **função Gen SOE**, o retorno reportado a eventos é definido pelo próprio algoritmo proprietário da função.

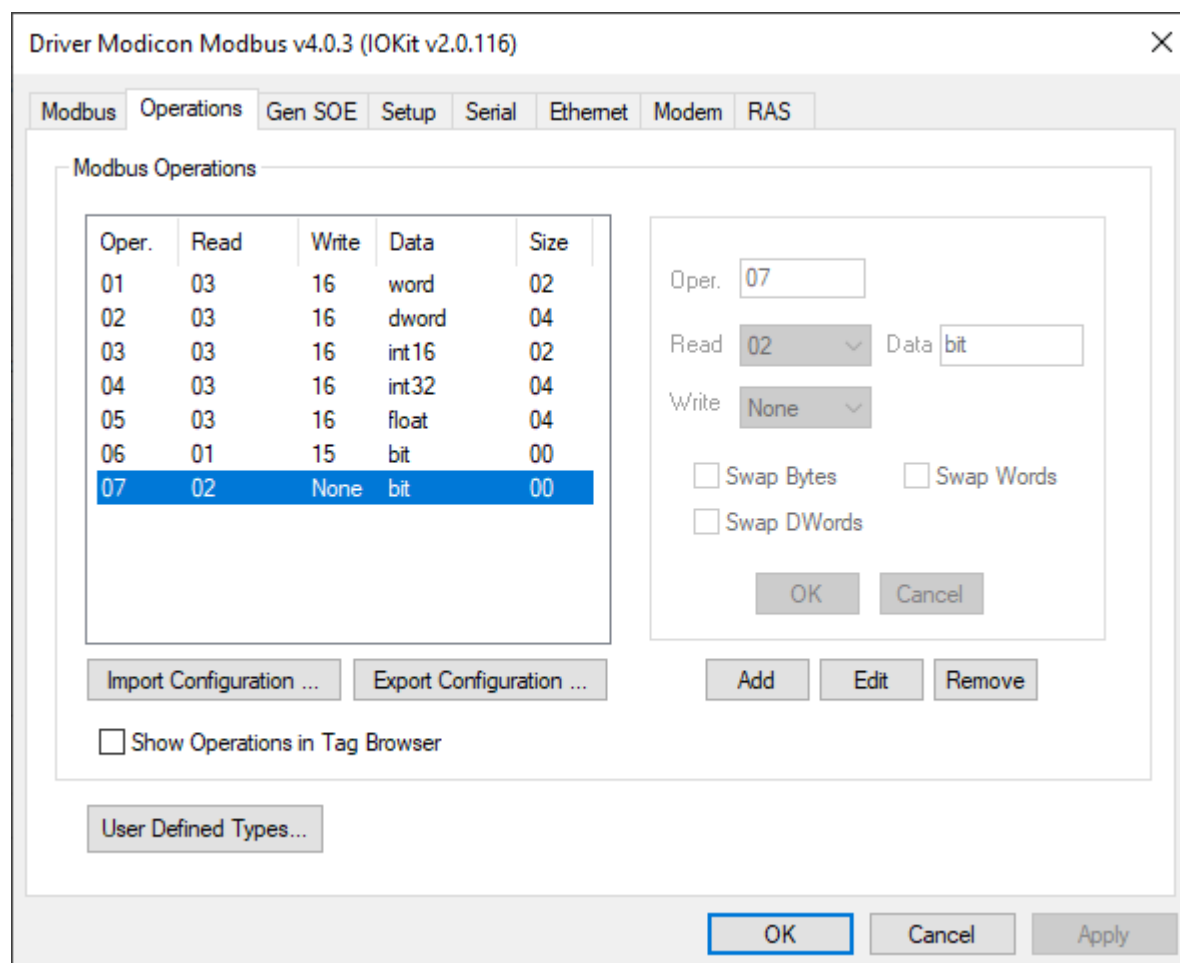
Para mais informações a respeito da configuração de Tags de Comunicação, consulte o tópico **Configurando um Tag de Comunicação**.

#### IMPORTANTE

Ao ler eventos de memória de massa em Tags reportados a eventos no **E3**, desabilite a banda morta no Tag (propriedade **EnableDeadBand** configurada como Falso) e também no objeto Histórico associado (propriedade **DeadBand** igual a zero), para evitar a perda de eventos com valores próximos. Também é importante desabilitar o histórico por varredura (no **E3**, propriedade **ScanTime** igual a zero). Com isto, garante-se que novos eventos só são armazenados através do método **WriteRecord**, executado no evento **OnRead** do Tag, evitando a duplicação de eventos.

## Importação e Exportação de Operações

A importação e exportação de operações deve ser realizada na aba **Operations** do Driver, clicando em **Import Configuration** ou em **Export Configuration**, conforme mostrado na figura a seguir.



Opções de importação e exportação de operações

Estas opções tornam possível importar e exportar a configuração de operações mostrada no quadro **Modbus Operations** para arquivos INI.

Nas versões anteriores à 2.00 deste Driver, a configuração das operações era executada no arquivo modbus.ini, que era carregado no momento da inicialização do objeto Driver. Os arquivos modbus.ini destas versões antigas ainda podem ser carregados na versão atual do Driver, utilizando a opção de importação.

### NOTA

As operações do Driver eram chamadas de **Funções do Driver** nas versões iniciais. Esta denominação foi posteriormente alterada para **Operações do Driver** devido a casos observados em que havia confusão com as **Funções do Protocolo**.

## Importação

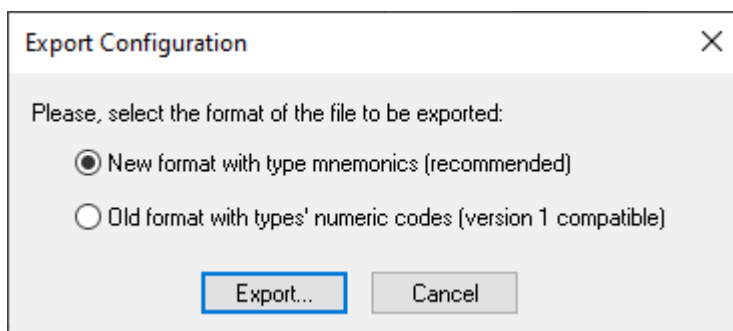
A importação de arquivos de configuração é bastante simples. Clique em **Import Configuration** e selecione o arquivo INI. O Driver deve carregar as configurações de operações, que aparecem imediatamente no quadro **Modbus Operations**. Este Driver permite a importação de arquivos gerados pelas suas versões anteriores.

## Exportação

A exportação de arquivos de configuração de operações pode ser realizada para compartilhar a mesma configuração de operações em diferentes objetos Driver, bem como para a eventual realização de cópia de segurança (*backup*) da configuração de operações de um determinado equipamento.

Outra possível utilidade é a exportação das configurações para um arquivo modbus.ini compatível com versões anteriores do Driver, permitindo carregar as configurações nesta versão anterior. Trata-se de uma prática pouco recomendada mas que, caso seja inevitável em aplicações legadas, requer as considerações feitas a seguir.

Ao clicar em **Export Configuration**, abre-se uma janela com duas opções, conforme a figura a seguir.



**Opções de exportação**

Nesta janela deve-se selecionar entre exportar conforme o novo padrão (**New format with type mnemonics**), com os tipos de dados exibidos sendo definidos com **Strings** (mnemônicos), ou no formato antigo (**Old format with types' numeric codes**), em que os tipos de dados eram identificados por um valor numérico, correspondente à posição em que apareciam na caixa de seleção **Data** na aba **Operations**.

O formato novo é mais legível, facilitando a depuração, e é utilizado nas versões mais recentes deste Driver, sendo a opção mais recomendada.

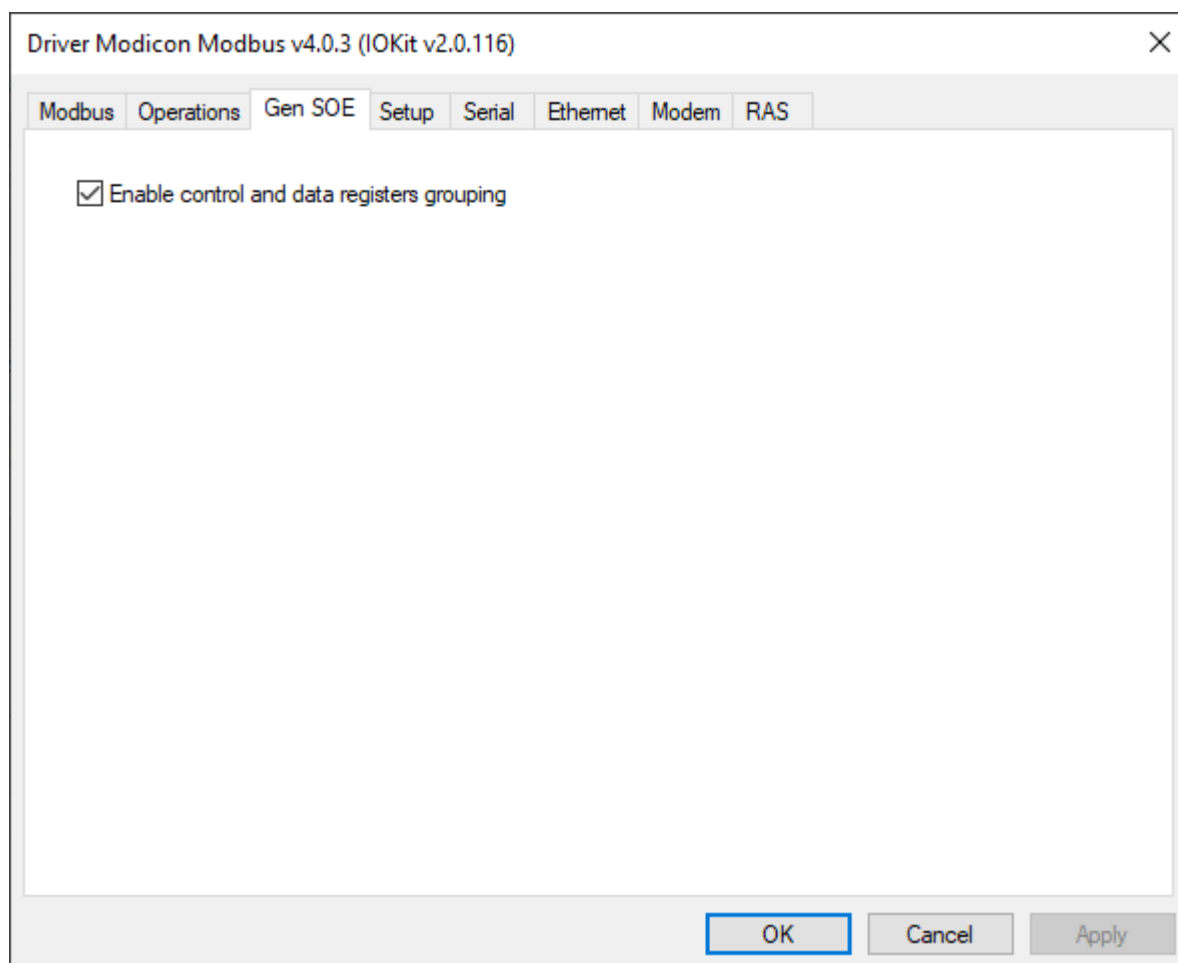
Já o formato antigo deve ser selecionado somente se for imprescindível exportar para versões anteriores à versão 2.08 deste Driver.

Note que, para exportar com sucesso arquivos modbus.ini a serem carregados em versões anteriores à versão 2.00, as operações não podem definir nenhum tipo de dados novo que não tenha sido implementado pela versão de destino, e nem tampouco ter operações que utilizem o parâmetro **Use bit mask**, caso contrário a importação pode falhar.

De maneira geral, recomenda-se evitar a exportação de configurações para versões anteriores, priorizando sempre a atualização do Driver.

## Aba Gen SOE

O objetivo desta aba é concentrar opções de configuração para o **Algoritmo de Leitura de SOE Genérico da Elipse Software**. A figura a seguir mostra a opção disponível nesta aba.



**Aba Gen SOE**

A única opção de configuração atualmente disponível está descrita a seguir:

- **Enable control and data registers grouping (padrão Verdadeiro):** Habilita o agrupamento de registros de controle e registros de dados, de forma a realizar o mínimo de leituras possível. Se esta opção estiver habilitada, o Driver inicia a leitura de tabelas já tentando ler o máximo de registros permitidos pelo protocolo, tanto registros de controle como de dados, e se possível já lendo a tabela inteira em uma única leitura. Tal procedimento em geral otimiza a varredura dos Tags da mesma forma que os Superblocos, pois o tempo demandado para a leitura de blocos grandes é em geral bem menor do que o tempo necessário para realizar várias leituras da mesma quantidade de dados, embora isto possa depender do CLP. Os CLPs ATOS não permitem a leitura agrupada dos registros de controle e de estruturas de dados, exigindo que esta opção seja desabilitada.

## Configuração em Modo Offline

As configurações do Driver também podem ser acessadas em tempo de execução se o Driver for iniciado no modo **Offline**, conforme explicado no tópico **Documentação das Interfaces de Comunicação**, utilizando-se os parâmetros de tipo **String** mostrados na tabela a seguir.

### Parâmetros disponíveis

PARÂMETRO	TIPO DE DADOS
<b>ModiconModbus.ModbusMode</b>	Inteiro: <ul style="list-style-type: none"> <li><b>0</b>: Modbus RTU</li> <li><b>1</b>: Modbus ASCII</li> <li><b>2</b>: Modbus TCP</li> </ul>
<b>ModiconModbus.Olderaddr</b>	Booleano (0 ou 1): <ul style="list-style-type: none"> <li><b>0</b>: Dado é endereçado a partir de 0 (zero)</li> <li><b>1</b>: Dado é endereçado a partir de 1 (um)</li> </ul>
<b>ModiconModbus.UseDefaultSlaveAddress</b>	Booleano (0 ou 1)
<b>ModiconModbus.DefaultSlaveAddress</b>	Inteiro sem sinal
<b>ModiconModbus.UseSwapAddressDelay</b>	Booleano (0 ou 1)
<b>ModiconModbus.SwapAddressDelay</b>	Inteiro, com o intervalo de <i>delay</i> em milissegundos. <b>NOTA:</b> Opção obsoleta e mantida por compatibilidade. Para aplicações novas, use a opção <b>Inter-frame delay</b> da aba <b>Serial</b> do <b>IOKit</b>
<b>ModiconModbus.WaitSilenceOnError</b>	Booleano (0 ou 1)
<b>ModiconModbus.EnableCMSAddressing</b>	Booleano (0 ou 1)
<b>ModiconModbus.EnCustomizeMaxPDUSize</b>	Booleano (0 ou 1)
<b>ModiconModbus.MaxPDUSize</b>	Inteiro
<b>ModiconModbus.ConfigFile</b>	<b>String</b> contendo o arquivo de configuração com as operações do Driver. Este arquivo pode ser exportado e importado na aba <b>Operations</b> da janela de configurações do Driver
<b>ModiconModbus.EnableReconnectAfterTimeout</b>	Booleano (0 ou 1): <ul style="list-style-type: none"> <li><b>0</b>: O <i>time-out</i> não gera desconexão do meio físico</li> <li><b>1</b>: Em caso de <i>time-out</i>, quando em meio físico Ethernet, o Driver promove a desconexão e reconexão do meio físico</li> </ul>
<b>ModiconModbus.UserTypesConfigFile</b>	<b>String</b> de configuração dos <b>tipos de dados definidos pelo usuário</b> (estruturas). Trata-se do mesmo arquivo de configuração que pode ser acessado na janela de configuração do Driver ( <b>User-Defined Data Types</b> )
<b>ModiconModbus.EnableGenSOERegGrouping</b>	Booleano (0 ou 1): <ul style="list-style-type: none"> <li><b>0</b>: O algoritmo de leitura de eventos primeiro lê os registros de controle e em seguida os dados de eventos</li> <li><b>1</b>: A leitura de SOE genérico é agrupada ao máximo, com a primeira leitura lendo não só os registradores de controle mas também o máximo de eventos possíveis</li> </ul>

PARÂMETRO	TIPO DE DADOS
<b>ModiconModbus.ShowOperationsInTagBrowser</b>	Booleano (0 ou 1): <ul style="list-style-type: none"> <li>• <b>0:</b> O Tag Browser mostra os modelos de Tags configurados por <b>Strings</b> (comportamento padrão)</li> <li>• <b>1:</b> O Tag Browser mostra os modelos de Tags configurados numericamente (comportamento legado)</li> </ul>

Para mais informações sobre a configuração **Offline** em tempo de execução, consulte o tópico **Documentação das Interfaces de Comunicação**.

## Configurando Tags

Este tópico descreve a configuração dos diversos tipos de Tags suportados por este Driver. Os Tags são divididos em duas categorias, descritas nos seguintes tópicos:

- **Configurando um Tag de Comunicação**
- **Configurando Tags Especiais**

### Configurando um Tag de Comunicação

Os Tags de Comunicação do Driver são os Tags que permitem a comunicação com os equipamentos.

Os Tags de Comunicação permitem ler e escrever em registros Modbus nos equipamentos escravos, usando as **funções do protocolo Modbus**, ou mesmo **funções especiais**. Este Driver não faz distinção entre Tags Bloco e Tags simples, no caso de Tags de Comunicação, ou seja, os Tags de Comunicação funcionam da mesma forma que Tags Bloco com apenas um Elemento.

Os dados são lidos do equipamento utilizando os formatos suportados pelo protocolo, ou seja, registros de valores inteiros de 16 bits, bytes ou conjuntos de bits, dependendo da função do protocolo utilizada. Para maiores informações sobre as funções do protocolo, consulte a especificação no *site oficial do protocolo*.

Estes Tags podem ser configurados de duas formas, descritas nos tópicos a seguir:

- **Configuração por Strings:** Este é o método mais novo que pode ser utilizado com o **Elipse E3**, **Elipse Power** e **Elipse OPC Server** e recomendado para novos projetos. Não funciona com o **Elipse SCADA**
- **Configuração Numérica:** Método antigo utilizado no **Elipse SCADA**



## Configuração por Strings

A configuração por **Strings** de **Tags de Comunicação** é realizada usando os campos **Dispositivo** e **Item** de cada Tag.

Este novo método de configuração não funciona no **Elipse SCADA**, que usa o antigo método de **configuração numérica** (parâmetros *N* e *B*).

Os parâmetros *N* e *B* não são usados na configuração por **Strings** e devem ser deixados em 0 (zero).

A configuração por **Strings** torna a configuração dos Tags mais legível, facilitando a configuração e manutenção das aplicações.

### Leitura em Bloco

Os Tags configurados por **Strings** podem ser Tags simples ou Tags Bloco, com os campos **Dispositivo** e **Item** tendo a mesma sintaxe.

#### NOTA

O serviço de agrupamento de Tags realizado pelos **Superblocos** não está disponível para os Tags que utilizam a configuração por **Strings**. Se houver necessidade de otimização através da leitura de superblocos, os Tags da aplicação devem utilizar somente a **Configuração Numérica**.

### Campo Dispositivo

No campo **Dispositivo**, o *Slave Id* (endereço identificador do equipamento) deve ser fornecido como um número entre 1 (um) e 255 seguido de dois pontos, como por exemplo "1:", "101:", "225:", etc.

Cabe lembrar que, no protocolo Modbus, o *Slave Id* 255 é reservado para *broadcast*, o que só faz sentido ser usado para operações de escrita, pois não há retorno dos equipamentos, ou ocorreriam conflitos.

Opcionalmente, o *Slave Id* pode aparecer no início do campo **Item**, explicado a seguir, e neste caso o campo **Dispositivo** deve ser mantido vazio. Esta opção é detalhada a seguir.

### Campo Item

No campo **Item** devem ser fornecidas todas as demais informações de endereçamento e da operação a ser realizada pelo Tag, com a seguinte a sintaxe:

```
<espaço de endereçamento><endereço>[.<tipo><tam. do tipo>][.<byte order>][/bit]
```

Onde os parâmetros entre colchetes são opcionais.

Como já mencionado na seção anterior, pode-se opcionalmente adicionar o *Slave Id* no início do campo **Item**, como no exemplo a seguir:

```
<slave id>:<espaço de endereçamento><endereço>[.<tipo><tam. do tipo>][.<byte order>][/bit]
```

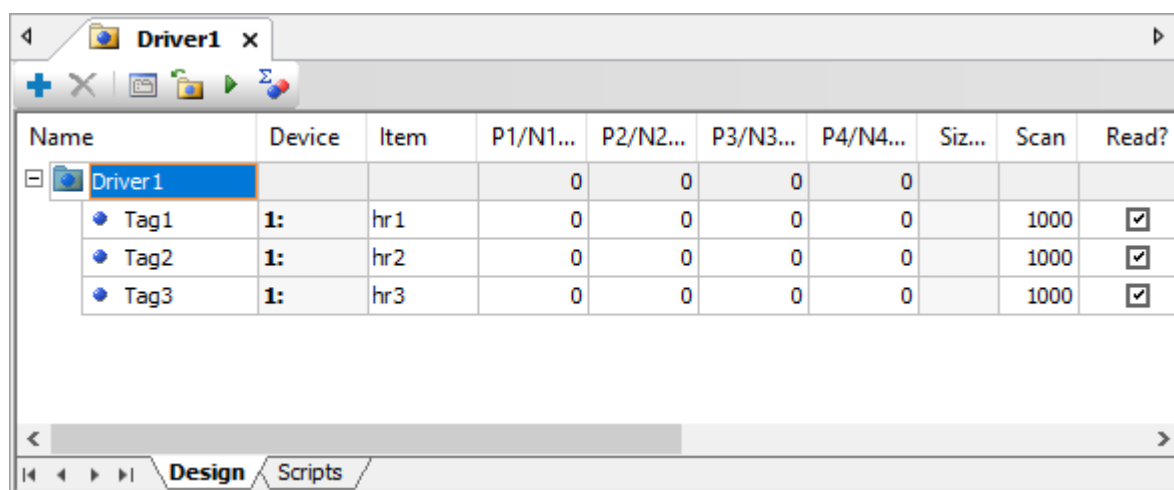
Neste caso, como já explicado, deve-se deixar o campo **Dispositivo** vazio.

Os exemplos a seguir mostram os casos de uso mais comuns (note que as aspas duplas não devem ser adicionadas no supervisório):

1. Leitura ou escrita de *Holding Registers* (funções **03** e **16**) de endereço 100 do equipamento com *Id* 1 e com *Slave Id* no campo **Dispositivo**:

- a. **Dispositivo:** "1:"
  - b. **Item:** "hr100"
2. Leitura ou escrita de *Holding Register* (funções **03** e **16**) de endereço 120 do equipamento com *Id* 3 e com Slave Id no campo **Item**:
    - a. **Dispositivo:** "" (**String** vazia)
    - b. **Item:** "3:hr120"
  3. Leitura ou escrita de *Coil* (funções **01** e **15**) de endereço 65535 (FFFFh) com *Slave Id* 4, aqui especificado no campo **Item**:
    - a. **Dispositivo:** "" (**String** vazia)
    - b. **Item:** "4:cl&hFFFF" (ou opcionalmente "4:cl65535")

A figura a seguir mostra exemplos de Tags do **E3** configurados por **Strings**.



Name	Device	Item	P1/N1...	P2/N2...	P3/N3...	P4/N4...	Siz...	Scan	Read?
Driver 1			0	0	0	0			
Tag1	1:	hr1	0	0	0	0		1000	<input checked="" type="checkbox"/>
Tag2	1:	hr2	0	0	0	0		1000	<input checked="" type="checkbox"/>
Tag3	1:	hr3	0	0	0	0		1000	<input checked="" type="checkbox"/>

Tags configurados por Strings

## Campos Obrigatórios e Opcionais

Os campos **obrigatórios** para todos os Tags são enumerados a seguir e detalhados individualmente mais adiante neste mesmo tópico:

1. **Espaço de endereçamento:** Mnemônico que define o conjunto de funções de leitura e escrita do protocolo a utilizar (veja a **tabela** em seção específica, mais adiante neste mesmo tópico).
2. **Endereço:** Valor numérico identificando o endereço do item (registrador ou bit) a ler ou escrever dentro do espaço de endereçamento definido. Os valores podem ser fornecidos em decimal, hexadecimal ou octal. Para valores em decimal não é preciso adicionar prefixo, ou pode ser usado o prefixo "&d". Para valores em hexadecimal, adicione o prefixo "&h" (por exemplo, "&hFFFF"). Já para octal, use o prefixo "&o" (por exemplo, "&o843"). Este endereço pode ter um deslocamento (*offset*) em relação ao endereço realmente enviado no *frame* de comunicação, o que depende da convenção utilizada pelo fabricante. Em caso de dúvidas sobre as convenções de endereçamento, consulte o tópico **Dicas de Endereçamento (Modbus Convention)**. Em especial, verifique se o equipamento implementa o *offset* padrão do *Data Model* do protocolo (veja as opções de **Data Address Model Offset** na **aba Modbus**).

Já os campos enumerados a seguir são **opcionais**, usados para extensões ao protocolo padrão ou para compatibilidade com equipamentos que não sejam plenamente aderentes ao protocolo (são também melhor detalhados mais adiante):

1. **Tipo:** Define como os bytes da área de dados do *frame* de comunicação devem ser interpretados. Se for omitido, são usados os tipos padrão definidos pelo protocolo para as funções em uso, ou seja, **Word** para funções de acesso a registradores e **Bit** para funções de acesso a dados digitais (*Coils* e *Discrete Inputs*). Consulte a **tabela de mnemônicos** dos tipos suportados, mais adiante neste tópico.
2. **Tamanho do tipo:** É necessário especificar este campo apenas para os tipos de tamanho variável, como **BCD** e **String**. O valor numérico indica o tamanho do tipo em bytes.
3. **Byte order:** Mnemônico indicando o ordenamento dos bytes dos valores numéricos. Se omitido, é usado o padrão do protocolo, com os bytes mais significativos vindo primeiro no *frame* de comunicação, o chamado *big endian*. Veja mais informações na seção específica sobre esta opção, mais adiante neste tópico.
4. **Bit:** Permite retornar um bit específico de um valor inteiro, e obviamente só faz sentido em funções Modbus que retornem valores inteiros (**Words**). Em geral recomenda-se não usar este recurso, dando preferência ao mapeamento de bits do supervisorio. O bit 1 (um) é o menos significativo e, quanto maior o valor, mais significativo é o bit. O valor máximo permitido obviamente depende do tamanho do tipo, sendo o máximo atualmente de 64 para tipos **Double**. Este campo corresponde à antiga opção **Use Bit Mask** da configuração numérica. Mais informações sobre esta opção podem ser obtidas no tópico **Aba Operations**.

## Exceções

As **funções Modbus 20 e 21** do protocolo Modbus, que acessam arquivos, utilizam sintaxe ligeiramente diferente da mostrada anteriormente:

```
fr<arquivo>.<registro>[.<tipo>[<tam. do tipo>]][.<byte order>][</bit>]
```

Exemplo:

- **Dispositivo:** "" (**String** vazia)
- **Item:** "1:fr4.101" (leitura do registro 101 do arquivo 4 no *Slave Id* 1)

Especificamente para a **função especial GenSOE** (*Elipse Generic SOE*):

```
elsoe<N>.<end. inicial>[.<tipo>[<tam. do tipo>]][.<byte order>][</bit>]
```

Especificamente para a **função especial SP SOE** (*Sepam SOE*), para leitura de todos os eventos:

```
spsoe<tipo do evento>.<end. inicial>[.<tipo>][.<byte order>][</bit>]
```

Especificamente para a **função especial SP SOE** (*Sepam SOE*), para a leitura de eventos de pontos específicos:

```
ptspsoe<tipo do evento>.<end. do evento>
```

Especificamente para a **função especial GE SOE** (*GE PAC RX7 SOE*):

```
gesoe<tipo do tag + índice do ponto>.<end. base da pilha>
```

Consulte os tópicos específicos sobre as funções especiais mencionadas anteriormente para mais informações sobre a configuração dos respectivos Tags usando **Strings**.

## Espaço de Endereçamento

Ao invés de definir explicitamente as funções Modbus ou especiais de leitura e escrita a serem utilizadas, como ocorre na antiga **configuração numérica** e seu conceito de **operações**, na configuração por **Strings** o usuário define um *espaço de endereçamento* através de um dos mnemônicos listados na tabela a seguir, já associado a funções pré-definidas do protocolo e seus respectivos tipos nativos de dados.

## Espaços de endereçamento e mnemônicos

ESPAÇO DE ENDEREÇAMENTO	MNEMÔNICO	TIPO NATIVO	FUNÇÃO	COMENTÁRIOS
<b>Holding Register</b>	hr	<b>Word</b> (16 bits)	Funções <b>03</b> e <b>16</b>	As funções <b>03</b> e <b>16</b> são as mais usadas do protocolo (Modbus classe 0)
<b>Single Holding Register</b>	shr	<b>Word</b> (16 bits)	Funções <b>03</b> e <b>06</b>	A função <b>06</b> escreve nos mesmos registradores da função <b>16</b> , a diferença é que a função <b>16</b> pode escrever em Blocos, enquanto a função <b>06</b> escreve em apenas um registro por vez, mas com menor <i>overhead</i>
<b>Coil</b>	cl	<b>Bit</b>	Funções <b>01</b> e <b>15</b>	
<b>Single Coil</b>	scl	<b>Bit</b>	Funções <b>01</b> e <b>05</b>	A função <b>05</b> escreve nos mesmos registros da função <b>15</b> , porém não pode escrever em Blocos, portanto com menor <i>overhead</i>
<b>Discrete Input</b>	di	<b>Bit</b>	Funções <b>02</b> e <b>None</b> (somente leitura)	
<b>Input Register</b>	ir	<b>Word</b> (16 bits)	Funções <b>04</b> e <b>None</b> (somente leitura)	
<b>Exception Status</b>	es	<b>Byte</b>	Funções <b>07</b> e <b>None</b> (somente leitura)	
<b>File Register</b>	fr	<b>Word</b> (16 bits)	Funções <b>20</b> e <b>21</b>	
<b>ABB MGE 144 - Leitura de Memória de Massa</b>	abbmge	<b>Word</b> (16 bits)	Funções <b>65 03</b> e <b>None</b> (somente leitura)	
<b>ABB MGE 144 - Reset</b>	abbmge.rst	Não usado	Função de leitura <b>None</b> e de escrita <b>65 01</b>	
<b>ABB MGE 144 - Zera Memória de Máximos e Mínimos</b>	abbmge.rstmxm	Não usado	Função de leitura <b>None</b> e de escrita <b>65 02</b>	
<b>GE PAC RX7 SOE - Leitura</b>	gesoe	<b>GE_events</b>	Função de leitura <b>GE SOE</b> e de escrita <b>None</b>	
<b>SEPAM SOE Events</b>	spsoe	<b>SP_events</b>	Função de leitura <b>SP SOE</b> e de escrita <b>None</b>	Coleta do medidor e retorna ao Tag uma estrutura (tipo <b>SP_events</b> ) com eventos de quaisquer pontos (veja o tópico <b>SEPAM SOE</b> )

ESPAÇO DE ENDEREÇAMENTO	MNEMÔNICO	TIPO NATIVO	FUNÇÃO	COMENTÁRIOS
<b>SEPAM SOE Single Point Events</b>	ptspsoe	<b>Int32</b>	Função de leitura <b>SP SOE</b> e de escrita <b>None</b>	Coleta do medidor e retorna ao Tag um valor inteiro do campo <b>Edge</b> , relativo a eventos de um ponto específico (veja o tópico <b>SEPAM SOE</b> )
<b>Elipse Generic SOE</b>	elsoe	<b>Word</b> (16 bits)	Função de leitura <b>Gen SOE</b> (função Modbus <b>03</b> com algoritmos adicionais) e de escrita <b>16</b>	

## Tipos de Dados

Na tabela da seção anterior são listados os tipos de dados nativos do protocolo, conforme as funções Modbus utilizadas, bem como alguns tipos de dados específicos usados em **funções especiais** (não padrão do protocolo). Para Tags que retornem estes tipos de dados nativos, o parâmetro *Tipo de Dado* pode ser omitido da **String** do campo **Item**.

Caso seja necessário interpretar os dados nativos de outra forma, algo muito comum entre os equipamentos que usam Modbus, deve-se especificar o tipo de dados a ser usado, conforme explicado nesta seção.

A lista e o detalhamento de todos os tipos de dados suportados por este Driver pode ser consultada no tópico **Tipos de Dados Suportados**.

A tabela a seguir lista os mnemônicos usados no parâmetro *<tipo>* do campo **Item** para cada tipo de dados suportado, nativo do Driver, e também um *alias* ou nome alternativo.

### Tipos de dados suportados

TIPO	MNEMÔNICO	ALIAS
<b>Char</b>	char	ch
<b>Byte</b>	byte	by ou u8
<b>Int8</b>	int8	i8
<b>Int16</b>	int16	i16
<b>Int32</b>	int32	i32
<b>Word ou UInt</b>	word	u16
<b>DWord ou ULong</b>	dword	u32
<b>Float</b>	float	f
<b>Float_GE</b>	float_GE	fge
<b>Double ou Real</b>	double	d
<b>String</b>	string	s
<b>BCD</b>	BCD	bcd
<b>GenTime</b>	GenTime	gtm
<b>Sp_time</b>	Sp_time	sptm

TIPO	MNEMÔNICO	ALIAS
UTC64d	UTC64d	-
UTC32	UTC32	-

## Tipos de Dados Criados pelo Usuário

Além dos tipos de dados listados na tabela anterior, usuários podem também fornecer mnemônicos de tipos de dados criados pelo usuário ou estruturas (veja o tópico **Tipos de Dados Definidos pelo Usuário**).

Para que os tipos de dados definidos pelo usuário possam ser usados no campo **Item**, entretanto, é preciso que os nomes destes tipos de dados não se diferenciem apenas por maiúsculas e minúsculas, uma vez que o campo **Item** não leva em conta caixa alta ou baixa. Se isto ocorrer, o Driver não permite o uso destes tipos de dados no campo **Item** (veja o tópico **Tipos de Dados Definidos pelo Usuário**).

## Exemplos

- Leitura ou escrita de *Holding Registers* (funções **03** e **16**) de endereço 100 do equipamento com *Id* 1, interpretado como um **DWord**, com *Slave Id* no campo **Dispositivo**:
  - Dispositivo**: "1:"
  - Item**: "hr100.u32" ou "hr100.dword", ou se for conveniente usar hexadecimal, "hr&h64.u32"
- Leitura ou escrita de *Holding Registers* (funções **03** e **16**) de endereço 150 do equipamento com *Id* 3, interpretado como um **Float**, com *Slave Id* no campo **Item**:
  - Dispositivo**: "" (**String** vazia)
  - Item**: "3:hr150.f" ou "3:hr150.float" ou em hexadecimal "3:hr&h96.f"
- Leitura ou escrita de *Holding Registers* (funções **03** e **16**) de endereço 1500 do equipamento com *Id* 5, interpretado como um **Double**, com *Slave Id* no campo **Item**:
  - Dispositivo**: "" (**String** vazia)
  - Item**: "5:hr1500.d" ou "5:hr1500.double" ou ainda, se for conveniente endereçar em hexadecimal, "5:hr&h5DC.d"
- Leitura ou escrita de *Holding Registers* (funções **03** e **06**) de endereço 100 do equipamento com *Id* 5, interpretado como um **Word**, com *Slave Id* no campo **Item**:
  - Dispositivo**: "" (**String** vazia)
  - Item**: "5:shr100" ou "5:shr100.u16" ou "5:shr100.word". Note que, por ser um **Word**, o tipo de dados nativo do protocolo Modbus para *Holding Registers*, o tipo de dados pode ser omitido
- Leitura ou escrita de *Holding Registers* (funções **03** e **06**) de endereço 100 do equipamento com *Id* 5, interpretado como o **tipo de dados do usuário** chamado "mytype", com *Slave Id* no campo **Item**:
  - Dispositivo**: "" (**String** vazia)
  - Item**: "5:shr100.mytype"

**NOTA**

O espaço de endereçamento de *Holding Registers* no protocolo Modbus contém registros de 16 bits. Portanto, para a leitura de tipos de dados de 32 bits, como **DWord** ou **Float**, é necessário ler dois endereços de "hr" para cada Tag acessado. Da mesma forma, a leitura de um Tag do tipo **Double** exige a leitura de quatro endereços de *Holding Registers*. Pelos mesmos motivos, a leitura e escrita de Tags de "hr" de tipo **Byte** só pode ser realizada aos pares. No equipamento, cada endereço de *Holding Register* contém sempre dois bytes.

## Tamanho do Tipo de Dados

Os tipos de dados **BCD** e **String**, por possuírem tamanho variável, exigem a especificação do tamanho do tipo de dados (*type size*), em bytes, logo após o tipo.

Cabe lembrar que **são válidos apenas os tipos de dados 2 e 4** (2 e 4 bytes ou 4 e 8 dígitos) para os tipos de dados **BCD**. Exemplos:

1. Leitura ou escrita de *Holding Registers* (funções **03** e **16**) de endereço 100 do equipamento com *Id* 1, interpretado como uma **String** de 10 bytes (cinco registros "hr"), com *Slave Id* no campo **Dispositivo**:
  - a. **Dispositivo**: "1:"
  - b. **Item**: "hr100.s10"
2. Leitura ou escrita de *Holding Registers* (funções **03** e **16**) de endereço 100 do equipamento com *Id* 1, interpretado como **BCD** de oito dígitos (quatro bytes ou dois registros "hr"), com *Slave Id* no campo **Item**:
  - a. **Dispositivo**: "" (**String** vazia)
  - b. **Item**: "1:hr100.bcd4"

## Ordenamento de Bytes (Byte Order)

Como mostrado nas sintaxes da seção anterior, pode-se adicionar o parâmetro opcional *byte order* no campo **Item** dos Tags para especificar o ordenamento de bytes para equipamentos que não seguem o padrão do protocolo. Se o equipamento segue o ordenamento padrão do protocolo Modbus, este campo pode ser omitido.

Caso ocorram leituras de valores distorcidos, o que pode ser observado já nos primeiros testes com o equipamento, e se estes valores, convertidos para hexadecimal, se mostrarem corretos com a inversão da posição de alguns bytes, leia esta seção atentamente.

O protocolo Modbus utiliza por padrão o formato *big endian*, onde os valores são formatados com os bytes mais significativos vindo primeiro nos *frames* de comunicação. Este é o formato que este Driver usa como padrão. Existe, entretanto, um grande número de equipamentos no mercado que se utilizam de valores com outras combinações de ordenamento dos bytes.

Como exemplo, se o Driver lê um valor igual a "1234h" (ou "4660" em decimal), por padrão o Driver espera que o dado seja enviado com a sequência de bytes 12h e 34h. Se entretanto o equipamento usa o padrão inverso, chamado de *little endian*, é enviado primeiro o byte 34h e depois o 12h, e o Driver pode interpretá-lo como 3412h, ou 13330 em decimal, a não ser que os dois bytes tenham sua ordem invertida antes de sua interpretação.

No caso de valores de 32 bits pode também ocorrer de valores em **Word** permutados (*swapped*), porém com os bytes dentro dos valores em **Word** mantendo a ordem padrão. Por exemplo, o valor 12345678h pode vir como 56781234h. E há vários outros casos, com inúmeras combinações diferentes de ordenamento.

Para permitir a comunicação com estes equipamentos que não seguem o padrão do protocolo de ordenamento de bytes, o Driver permite ao usuário configurar Tags especificando o padrão a ser utilizado.

O parâmetro *byte order* corresponde às **opções de swap** das operações, utilizadas antigamente na **configuração numérica**, e pode ter os valores "b0", "b1", "b2", "b3", "b4", "b5", "b6", "b7", "alias" ou ainda "alias2" (veja a tabela a seguir).

Se o parâmetro *byte order* for omitido, o dado é interpretado seguindo o padrão do protocolo, o que equivale ao código "b0".

A tabela a seguir indica que operações de *swap* ou permutas (*Swap Bytes*, *Swap Words* e *Swap DWords*) são realizadas para cada mnemônico de ordenamento (de "b0" até "b7").

#### Operações de permuta ou swap

	SWAP BYTES	SWAP WORDS	SWAP DWORDS	ALIAS	ALIAS 2 (SWAPS)	ORDENAMENTO *
<b>b0</b>				msb	-	by7 by6 by5 by4 by3 by2 by1 by0
<b>b1</b>	X			-	sb	by6 by7 by4 by5 by2 by3 by0 by1
<b>b2</b>		X		-	sw	by5 by4 by7 by6 by1 by0 by3 by2
<b>b3</b>	X	X		-	sb.sw	by4 by5 by6 by7 by0 by1 by2 by3
<b>b4</b>			X	-	sdw	by3 by2 by1 by0 by7 by6 by5 by4
<b>b5</b>	X		X	-	sb.sdw	by2 by3 by0 by1 by6 by7 by4 by5
<b>b6</b>		X	X	-	sw.sdw	by1 by0 by3 by2 by5 by4 by7 by6
<b>b7</b>	X	X	X	lsb	sb.sw.sdw	by0 by1 by2 by3 by4 by5 by6 by7

\* 64 bits (onde "by0" é "lsb" e "b7" é "msb")

Ou seja, "b0" não realiza nenhuma operação de permuta nos bytes de dados, mantendo o ordenamento de bytes original recebido do equipamento, equivalente a não selecionar nenhuma **opção de permuta na aba Operations** da antiga **configuração numérica**.

Já "b1" realiza a permuta dos bytes, dois a dois, ou seja, ao receber um **Word** (inteiro sem sinal de 16 bits) com o valor em hexadecimal 0102h, o valor que é retornado ao Tag é 0201h, com os bytes trocados. Equivale à antiga opção **Swap Bytes**.

Já o código "b2" realiza a permuta de **Words**, ou seja, de duplas de bytes, dois a dois, o que obviamente só afeta dados de 32 bits ou maiores. Tem o mesmo efeito de selecionar a opção **Swap Words** na configuração numérica. Como exemplo, se for recebido do equipamento o valor 01020304h, o valor utilizado para os Tags da aplicação é 03040102h.

Já se for usado o código "b3", tem-se a troca de bytes e **Words**, equivalente às antigas opções **Swap Bytes** e **Swap Words** habilitadas simultaneamente. Neste caso, o valor 01020304h se torna 04030201h.



Da mesma forma, o código "b4" realiza a troca de **DWords** entre si em valores de 64 bits, como a antiga opção **Swap DWords** da **configuração numérica**, ou seja, o valor 1122334455667788h é interpretado como 5566778811223344h. E assim por diante para os demais códigos.

As duas últimas colunas da tabela especificam apelidos ou *aliases* que o usuário pode usar para facilitar a legibilidade, ou seja, ao invés de usar o código "b0", pode ser utilizado "msb". Ao invés do código "b6", pode-se utilizar o apelido "sw.sdw", e assim por diante.

## Como Selecionar a Opção de Ordenamento Correta?

Em muitos casos a documentação do equipamento especifica qual o ordenamento utilizado, ou como configurá-lo (há equipamentos que permitem esta configuração).

Nos casos em que a documentação é omissa, deve-se contatar o suporte do fabricante.

Caso não seja possível obter-se informações confiáveis, deve-se testar empiricamente, analisando os valores retornados, em hexadecimal, comparando com os valores esperados e observando se existem inversões de ordenamento que possam explicar as diferenças.

Podem ocorrer basicamente três situações:

1. Caso o equipamento forneça dados usando o ordenamento de bytes padrão do protocolo Modbus (*big endian* ou *Motorola*), com os bytes mais significativos vindo antes, deve-se omitir este parâmetro, ou defini-lo como "b0". Esta é a situação mais comum.
2. Caso o equipamento use outro padrão de ordenamento de bytes, com os bytes menos significativos vindo antes (*little endian*), é necessário habilitar-se todas as opções de permuta referentes ao tipo de dados usado, o que corresponde ao mnemônico "b7".
3. No caso menos comum, há equipamentos que usam ordenamentos de bytes diferentes para tamanhos de dados diferentes, fornecendo por exemplo o byte mais significativo de cada **Word** primeiro, porém o **Word** menos significativo de cada **DWord** primeiro. Portanto, é preciso avaliar em qual caso cada opção de permuta precisa ser habilitada, de maneira a converter o valor retornado pelo equipamento para o formato *big endian* padrão do protocolo.

### NOTA

As opções de permuta citadas não têm efeito para tipos de dados **Bit** ou tipos com tamanho de oito bits (**Byte**, **Char** e **Int8**). A permuta ocorre dentro de cada tipo de dado, ou seja, a opção **Swap Words** não tem efeito para tipos de dados de 16 bits, assim como a opção **Swap DWords** não tem efeito para tipos de dados de 32 bits. Os tipos de dados **BCD** também não permitem operações de permuta.

O tópico **Dúvidas Mais Frequentes** lista alguns casos conhecidos, já observados nos atendimentos de suporte. Exemplos:

1. Leitura ou escrita de *Holding Registers* (funções **03** e **16**) de endereço 1500 do equipamento com *Id* 5, interpretado como um **Double** sem inversão de bytes, com *Slave Id* no campo **Item**:
  - a. **Dispositivo**: "" (**String** vazia)
  - b. **Item**: "5:hr1500.d", ou "5:hr1500.double" ou ainda "5:hr1500.d.b0"
2. Leitura ou escrita de *Holding Registers* (funções **03** e **16**) de endereço 1500 do equipamento com *Id* 5, interpretado como um **Double** com o byte menos significativo de cada **Word** vindo antes e com *Slave Id* no campo **Item**:
  - a. **Dispositivo**: "" (**String** vazia)

- b. **Item:** "5:hr1500.d.b1" ou "5:hr1500.double.b1", ou ainda "5:hr1500.double.sb"
- 3. Leitura ou escrita de *Holding Registers* (funções **03** e **16**) de endereço 1500 do equipamento com *Id* 5, interpretado como um **Double** com os bytes menos significativos vindo antes (*little endian*) e com *Slave Id* no campo **Item**:
  - a. **Dispositivo:** "" (**String** vazia)
  - b. **Item:** "5:hr1500.d.b7" ou outras variações, como "5:hr1500.d.lsb" e "5:hr1500.d.sb.sw.sdw"

## Tags Especiais do Driver

Além dos Tags já descritos, pode-se configurar **Tags Especiais** do Driver usando **Strings**, descritos em mais detalhes em tópicos específicos (clique no item desejado para mais informações).

Tags especiais

DEVICE	ITEM	OPERAÇÃO
	<b>ForceWaitSilence</b>	Escrita
<b>&lt;slave id&gt;:</b>	<b>LastExceptionCode</b>	Leitura ou escrita

## Configuração Numérica

A configuração numérica é realizada através dos parâmetros *N* e *B* dos **Tags de Comunicação**, não utilizando os campos **Dispositivo** e **Item** disponíveis no **Elipse E3** e **Power**, que devem ser deixados em branco.

Este método de configuração deve ser utilizado no **Elipse SCADA** e em aplicações legadas. Em aplicações utilizando produtos mais novos, como o **Elipse E3**, **Elipse Power** ou **Elipse OPC Server**, recomenda-se utilizar a **configuração por Strings**.

Os **Tags de Comunicação** configurados numericamente referenciam as **operações** previamente configuradas na janela de configuração.

### Operações

Conforme já explicado no tópico **Aba Operations**, o Driver possui suporte a outros tipos de dados além dos tipos de dados nativos do protocolo. Por este motivo, foi criado o conceito de **Operação** no Driver.

Nas operações utilizando as funções Modbus que leem e escrevem bits, como as funções **1**, **5** e **15** do protocolo, o Driver mapeia sempre os valores binários de cada bit para os Elementos de Bloco, onde cada Elemento representa o valor de um bit específico.

As operações com tipos de dados de oito bits, como o tipo de dados **Byte**, implicam sempre, obviamente, na leitura de pelo menos dois bytes (um registro Modbus de 16 bits). Para prevenir surpresas para o usuário, o Driver exige que a escrita de dados de oito bits seja realizada sempre aos pares, ou seja, como escritas de Blocos com número par de Elementos. As operações devem ser referenciadas através dos parâmetros *N2/B2* dos Tags de Comunicação, conforme descrito a seguir.

### Parâmetros de Configuração dos Tags de Comunicação

A configuração a seguir se aplica tanto aos parâmetros *N* dos Tags de Comunicação quanto aos parâmetros *B* dos Tags Bloco de Comunicação.

- **N1/B1:** Endereço do equipamento escravo (CLP) na rede (*Slave Id*). Este endereço é usado em redes seriais e pode variar de 1 a 247. Pode-se ainda configurar este parâmetro com o valor 0 (zero). Com isto, este Tag trabalha em modo **Broadcast**, enviando a mensagem para todos os equipamentos escravos (CLP) que estiverem na rede. Em **Ethernet** (modo **Modbus TCP**), o endereço geralmente utilizado é apenas o endereço IP, porém o *Slave Id* pode ainda ser usado quando o endereço IP referencia um *gateway* ligado a uma rede de equipamentos (geralmente uma rede RS485, com Modbus RTU, usando *gateway* capaz de realizar a conversão de **Modbus TCP** para **Modbus RTU**).

#### NOTA

No modo **Broadcast** com *N1* igual a 0 (zero) não é possível realizar leituras, apenas escritas. Neste modo todos os equipamentos na rede são endereçados, recebendo o valor escrito e não retornando qualquer resposta, de forma a evitar conflitos na rede.

- **N2/B2:** Código da operação. Referencia uma operação adicionada na janela de configurações do Driver (veja o tópico **Aba Operations**).
- **N3/B3:** Parâmetro adicional. Este parâmetro em geral não é usado e pode ser deixado em 0 (zero). Só é usado em quatro situações:
  - **Funções Modbus 20 e 21:** No caso de operações que utilizem estas funções de acesso a arquivo (funções **20** e **21**), o parâmetro *N3/B3* especifica o arquivo a ser acessado.

- **Use Bit Mask:** Para Tags referenciando operações com a opção **Use Bit Mask** habilitada, o parâmetro *N3/B3* especifica o número do bit a ser acessado (veja o tópico **Aba Operations**).
- **Tipos de Dados Definidos pelo Usuário:** Para operações que usem estruturas, se o parâmetro *B3* for maior que 0 (zero), define o retorno de um *array* de blocos reportado a eventos, por meio de uma sequência de eventos **OnRead** do Tag (veja o tópico **Tipos de Dados Definidos pelo Usuário**).
- **Função Especial Gen SOE:** Em operações que utilizem a função especial de leitura **Gen SOE**, o parâmetro *N3/B3* indica o tamanho da tabela associada na memória do CLP ou dispositivo escravo, em número máximo de eventos comportados (veja o tópico **Algoritmo de Leitura de SOE Genérico da Elipse Software**).
- **N4/B4:** Endereço do registrador, variável ou bit no equipamento ou dispositivo escravo (CLP) em que se deseja ler ou escrever, conforme o mapa de registradores do equipamento (verifique o manual do equipamento). É importante ajustar corretamente a opção **Data Address Model Offset** (veja o tópico **Aba Modbus**) e verificar se a documentação do fabricante não usa *offsets* utilizados por equipamentos Modbus antigos, da chamada **Modbus Convention**.
- **Tamanho/Índice (somente Tags Bloco):** Cada Elemento de Bloco representa o valor de um dado de tipo definido na operação utilizada (parâmetro *N2/B2*). Note que o protocolo só suporta tipos de dados **Bit** ou **Word**. Desta forma, se a operação selecionar o tipo de dados **DWord** (32 bits) para cada Elemento de Bloco, o Driver necessita ler dois registros consecutivos do equipamento.

## Tags Especiais

Além dos Tags de Comunicação (Tags que referenciam operações), existem também Tags especiais para executar funções específicas do Driver. Estes Tags são descritos no tópico **Configurando Tags Especiais**.

## Dicas de Endereçamento (Modbus Convention)

No tópico **Configurando um Tag de Comunicação**, o endereçamento dos Tags (parâmetros *N4/B4* na **configuração numérica**) é descrito com base na especificação mais recente do protocolo Modbus (versão 1.1b). Entretanto, há equipamentos que ainda utilizam a antiga convenção de endereçamentos com *offsets* conhecida como **Modbus Convention**, que acrescenta *offsets* ao endereço. Este tópico explica como endereçar os Tags caso o mapa de registradores do equipamento ainda siga esta convenção antiga, originária da especificação inicial da Modicon, não mais incluída na especificação atual.

O endereço fornecido no Tag é enviado no *frame* de requisição do protocolo ao equipamento, acrescido ou não do *offset* padrão de 1 (um), requerido pelo Modbus Data Model especificado pelo protocolo, conforme configuração do campo **Data Model Offset** na aba **Modbus** da janela de configurações do Driver.

Além deste *offset* padrão de 1 (um), definido na norma Modbus atual (versão 1.1b), alguns fabricantes utilizam ainda o padrão antigo da Modicon, conhecido como **Modbus Convention**, com um *offset* que pode ser acrescentado ao endereço, cujo valor depende da função Modbus utilizada, ou mais especificamente, de qual espaço de endereçamento (*address space*) esta função acessava originalmente. Este *offset* adicional deve ser ignorado ao definir o endereço dos Tags neste Driver. Mais adiante são fornecidos alguns exemplos. A tabela a seguir lista os *offsets* utilizados pelo padrão **Modbus Convention**.

Offsets do padrão Modbus Convention

TIPO DE DADOS (STANDARD DATA MODEL)	FUNÇÃO MODBUS	OFFSET
Coils	01: Read Coils (0x)	00000
	05: Write Single Coil (0x)	
	15: Write Multiple Coils (0x)	
Discrete Inputs	02: Read Discrete Inputs (1x)	10000

TIPO DE DADOS (STANDARD DATA MODEL)	FUNÇÃO MODBUS	OFFSET
<b>Input Registers</b>	<b>04:</b> Read Input Registers (3x)	30000
<b>Holding Registers</b>	<b>03:</b> Read Holding Registers (4x) <b>06:</b> Write Single Register (4x) <b>16:</b> Write Multiple Registers (4x)	40000
<b>File Register (antiga Extended Memory file)</b>	<b>20:</b> Read General Reference (6x) <b>21:</b> Write General Reference (6x)	60000

Se o mapa de registradores do equipamento utiliza esta convenção, deve-se seguir este procedimento para determinar os endereços a serem atribuídos aos Tags, no campo **Item** na **configuração por Strings** ou aos parâmetros **N4** ou **B4** na **configuração numérica**:

1. Na aba **Modbus**, selecione a opção **Data is addressed from 1**.
2. Subtraia do endereço mostrado no manual do equipamento o *offset* mostrado na tabela anterior para a função Modbus utilizada. **DICA:** Remova o quinto dígito da direita para a esquerda.

Note que, em equipamentos que utilizam esta antiga convenção, pode-se determinar quais funções Modbus devem ser usadas para acessar cada registro ou bit através do *offset* empregado em seu endereço.

#### Exemplos

ENDEREÇO COM OFFSET (EQUIPAMENTO)	ENDEREÇO NO TAG (ITEM OU N4/B4)	FUNÇÃO MODBUS
<b>01234</b>	1234	<b>01:</b> Read Coils <b>05:</b> Write Single Coil <b>15:</b> Write Multiple Coils
<b>11234</b>	1234	<b>02:</b> Read Discrete Inputs
<b>31234</b>	1234	<b>04:</b> Read Input Registers
<b>41234</b>	1234	<b>03:</b> Read Holding Registers <b>06:</b> Write Single Register <b>16:</b> Write Multiple Registers
<b>45789</b>	5789	<b>03:</b> Read Holding Registers <b>06:</b> Write Single Register <b>16:</b> Write Multiple Registers
<b>65789</b>	5789	<b>20:</b> Read General Reference <b>21:</b> Write General Reference

## Partição Automática de Blocos

A partir da versão 2.00, o Driver Modbus passa a contar com o recurso de **Partição Automática de Blocos**. Com este recurso, o Driver passa a gerenciar sozinho a divisão de blocos maiores que os **limites do protocolo**. Assim, o usuário não precisa se preocupar em exceder o limite máximo de tamanho de blocos, pois o próprio Driver se encarrega de dividir os blocos nos tamanhos corretos no momento da comunicação com o equipamento, caso algum Tag Bloco exceda o tamanho máximo permitido.

A partir da versão 2.01, o Driver passou também a suportar **Leitura por Superblocos**. Com este recurso habilitado, o usuário não precisa mais agrupar variáveis em Tags Bloco objetivando melhoria de desempenho, sendo possível usar apenas Tags sem quaisquer prejuízos de performance. E como o algoritmo de Superblocos já leva em consideração o tamanho máximo de blocos permitido pelo protocolo, quando este recurso for utilizado o Driver também não necessita usar o recurso de Partição Automática.

Nos casos em que, devido à peculiaridades do equipamento (veja o tópico **Leitura por Superblocos**), não seja possível habilitar a propriedade **EnableReadGrouping** no **E3** ou **Elipse Power** (propriedade que habilita os Superblocos), ou caso se esteja utilizando o antigo **Elipse SCADA**, que não possui o recurso de agrupamento (Superblocos), é preciso contar com a Partição Automática de Blocos para que não seja necessário observar os limites do protocolo.

### IMPORTANTE

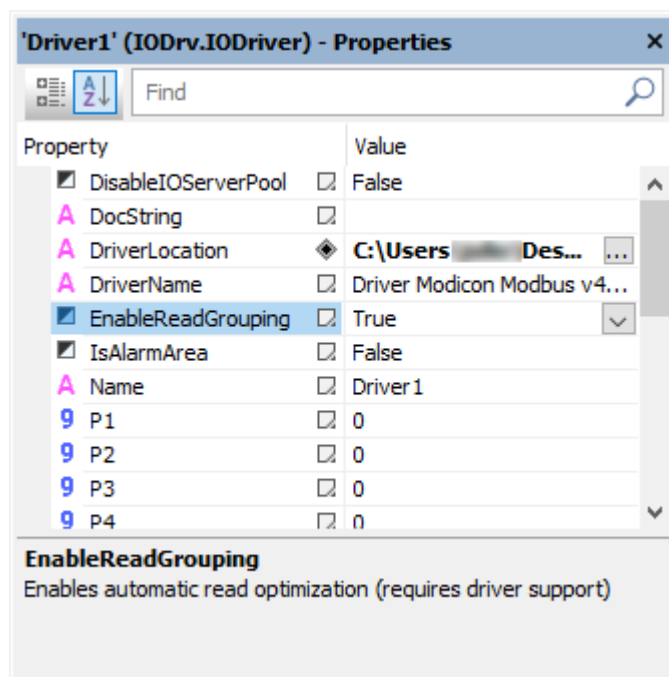
Tanto o agrupamento dos Superblocos no **E3** e no **Elipse Power** como a Partição Automática de Blocos do Driver requerem que o equipamento suporte os limites estabelecidos pelo Modbus padrão (veja o tópico **Limite Máximo para o Tamanho dos Blocos Suportado pelo Protocolo**). Há equipamentos, entretanto, que suportam limites inferiores. Para que a divisão automática de blocos e o próprio agrupamento dos Superblocos funcione nestes casos, a partir da versão 2.03 o Driver permite personalizar o limite máximo suportado para o PDU (*Protocol Data Unit*). Para isto, na janela de configurações do Driver, aba **Modbus**, habilite a opção **Customize Max. PDU Size** e configure o tamanho máximo de bytes suportado para o PDU pelo equipamento. Caso o equipamento possua limites diferentes para cada função, pode ser necessário realizar o agrupamento manualmente (veja o tópico **Leitura por Superblocos**).

O artigo *KB-23112* do **Elipse Knowledgebase** apresenta um resumo das questões relativas ao agrupamento de Tags e dimensionamento de blocos no Driver Modbus, discutidas neste e em outros tópicos (veja os tópicos **Leitura por Superblocos** e **Dicas de Otimização**).

## Leitura por Superblocos (Agrupamento)

A partir da versão 2.01, o Driver passa a suportar o recurso de **Leitura por Superblocos**. Este recurso é suportado pelo **E3** e pelo **Elipse Power**, podendo ser habilitado através da propriedade **EnableReadGrouping** do objeto Driver no Organizer. Com esta propriedade em Verdadeiro, o usuário não precisa se preocupar com o dimensionamento dos blocos.

Com este recurso torna-se possível, e em geral recomendável, criar aplicações contendo apenas Tags simples (Tags PLC no **Elipse SCADA**) sem prejuízo de desempenho, pois a otimização do agrupamento na leitura é feita automaticamente no momento da comunicação. A figura a seguir mostra a configuração da propriedade **EnableReadGrouping** no **E3** ou **Elipse Power**.



**Propriedade EnableReadGrouping**

O **Elipse SCADA** não possui suporte a Superblocos. O comportamento da leitura de Tags no **Elipse SCADA** é idêntico ao do **E3** e do **Elipse Power** quando a opção **EnableReadGrouping** estiver configurada como Falso. Em ambos os casos, o Driver conta com a **Partição Automática de Blocos**, podendo dividir blocos com tamanhos superiores ao **limite do protocolo** em blocos menores no momento da comunicação. Nestes casos, o usuário precisa levar em consideração o agrupamento ao definir os Tags da aplicação, como é visto mais adiante neste tópico.

### NOTAS

- O agrupamento automático é realizado com base nos Tags em *advise* da aplicação. Sempre que novos Tags entrarem em *advise* ou saírem de *advise*, o algoritmo de Superblocos redefine o agrupamento, ou seja, os Superblocos a serem lidos de forma automática, em tempo de execução, incluindo apenas os Tags que estiverem em *advise*
- Somente os Tags que utilizarem configuração numérica podem ser agrupadas pelo serviço de Superblocos. Os Tags que utilizem configuração por **Strings** não são agrupados pelo serviço de Superblocos

**IMPORTANTE**

Tanto o agrupamento dos Superblocos no **E3** como a **Partição Automática de Blocos** do Driver requerem que o equipamento suporte os limites estabelecidos pelo Modbus padrão (veja o tópico **Limite Máximo para o Tamanho dos Blocos Suportado pelo Protocolo**). Há equipamentos, entretanto, que suportam limites inferiores. Para que a divisão automática de blocos e o próprio agrupamento dos Superblocos funcione nestes casos, a partir da versão 2.03, o Driver permite personalizar o limite máximo suportado para o PDU (*Protocol Data Unit*). Para isto, na janela de configurações do Driver, aba **Modbus**, habilite a opção **Customize Max. PDU Size** e configure o tamanho máximo de bytes suportado para o PDU pelo equipamento. Caso o equipamento suporte limites diferentes para cada tipo de função, é necessário realizar o agrupamento manual (veja mais adiante neste tópico), observando os limites descritos na documentação do fabricante.

**Identificação de Equipamentos que não Suportam Agrupamento Automático (Superblocos)**

O algoritmo de Superblocos leva em conta os limites e espaços de endereçamento definidos pelo protocolo Modbus padrão. Nos casos de equipamentos que implementem o protocolo Modbus com pequenas variações, podem ser necessárias configurações avançadas adicionais para que seja possível utilizar o recurso de Superblocos, caso seu uso se mostre viável. Nestes casos, é necessário desabilitar o agrupamento automático (propriedade **EnableReadGrouping** configurada para Falso), realizando o agrupamento de forma manual. As seguintes condições podem tornar impossível a utilização de Superblocos, ou requerer configurações avançadas adicionais:

- Equipamentos que definem limites máximos de tamanho de bloco inferiores ao limite padrão do protocolo (**limite de 253 bytes para o PDU**). **Solução:** Ajuste a opção **Customize Max. PDU Size**, na aba **Modbus**

**NOTA**

Há equipamentos cujos limites de PDU variam conforme a função Modbus utilizada. Nesses casos, se for necessário usar funções com limites diferentes, também é preciso desabilitar os Superblocos (propriedade **EnableReadGrouping** configurada para Falso), agrupando os Tags manualmente (veja mais adiante neste tópico).

- Equipamentos com discontinuidades (intervalos de endereços não definidos intercalados com intervalos válidos) no mapa de registradores. **Solução:** Uma vez que é impossível informar ao algoritmo de Superblocos quais intervalos não podem ser inseridos em blocos, em geral não é possível usar Superblocos. Desabilite os Superblocos (propriedade **EnableReadGrouping** configurada para Falso) e agrupe os Tags manualmente
- Equipamentos que não suportam a leitura em blocos. **Solução:** Desabilite os Superblocos (propriedade **EnableReadGrouping** configurada para Falso) e defina Tags simples
- Equipamentos que só permitem a definição de blocos em endereços pré-determinados e com tamanhos fixos. **Solução:** Desabilite os Superblocos (propriedade **EnableReadGrouping** configurada para Falso) e defina Tags simples (Tags PLC no **Elipse SCADA**) ou Blocos de acordo com o especificado para o equipamento

**Agrupamento Manual**

Em geral, quanto maior for o agrupamento das variáveis em blocos, menor é o número de requisições de leitura necessárias para completar o ciclo de varredura (*scan*) dos Tags da aplicação, aumentando assim a velocidade de atualização dos Tags. Por este motivo, se não for possível usar o agrupamento automático (Superblocos), é preferível criar Tags Bloco contendo o maior número possível de variáveis ao invés de criar Tags simples (Tags PLC no **Elipse SCADA**).

Note que, devido ao recurso de **Partição Automática de Blocos**, não é necessário cuidar para que os limites máximos do protocolo sejam excedidos, desde que o equipamento suporte os **limites máximos padrão do protocolo**. Se o equipamento não suportar estes limites, mas definir limites fixos, válidos para todas as funções Modbus suportadas, deve-se configurar a opção **Customize Max. PDU Size**, na aba **Modbus**.



Caso o equipamento suporte limites diferentes para cada função suportada, pode ser inviável contar também com o particionamento automático. Nestes casos, o desenvolvedor da aplicação precisa levar em conta os limites do equipamento, e definir seus blocos cuidando para respeitar estes limites.

Para o agrupamento manual, o uso de **Tipos de Dados Definidos pelo Usuário** pode ampliar as possibilidades de agrupamento, por permitir reunir em um mesmo Tag Bloco variáveis do mesmo espaço de endereçamento, ou seja, uma mesma função Modbus, porém com tipos de dados diferentes (a estrutura definida pode ter elementos com tipos de dados diversos).

Para mais dicas, consulte o tópico **Dicas de Otimização**. O artigo *KB-23112* do **Eclipse Knowledgebase** apresenta um resumo das questões relativas ao agrupamento de Tags e dimensionamento de blocos no Driver Modbus, discutidas neste e em outros tópicos.

## Configurando Tags Especiais

Além dos **Tags de Comunicação**, este Driver suporta também alguns **Tags Especiais** que permitem à aplicação acionar recursos que não estão relacionados à leitura e escrita de dados. Ao contrário dos Tags de Comunicação, os Tags Especiais não referenciam **operações do Driver** na **configuração numérica**. Os tópicos a seguir detalham os Tags Especiais suportados:

- **Forçar Wait Silence**
- **Leitura do Código da Última Exceção**

### Forçar Wait Silence

Tag Especial utilizado para descartar todos os dados pendentes da comunicação até encontrar um *time-out*, indicando que não há mais dados a serem recebidos.

Este mesmo serviço pode ser configurado na aba **Modbus**, para ocorrer sempre que algum erro de comunicação for detectado. Com este Tag, entretanto, é possível executar o serviço a qualquer momento pela aplicação.

Este Tag Especial é executado através de um comando de escrita de Tag. O valor em si, escrito no Tag, é ignorado.

### Configuração por Strings

- **Dispositivo:** Não usado, deve-se deixar este campo vazio
- **Item:** "ForceWaitSilence"

### Configuração Numérica

- **N1:** Não usado, pode ser deixado em zero
- **N2:** 9001
- **N3:** Não usado, pode ser deixado em zero
- **N4:** Não usado, pode ser deixado em zero
- **Valor:** Não usado, pode ser deixado em zero

## Leitura do Código da Última Exceção

Conforme já mencionado neste Manual, os Tags Especiais para leitura do código da última exceção são utilizados para ler o último código de exceção enviado por um determinado equipamento escravo.

Tais códigos são armazenados automaticamente pelo Driver em registradores internos, que podem ser acessados por meio deste Tag. Além disto, a cada comunicação bem sucedida com determinado equipamento onde nenhuma exceção for retornada, o Driver zera automaticamente o registrador associado.

### Códigos de Exceção

Os códigos de exceção são usados pelo dispositivo escravo (CLP) para informar uma falha ao executar uma determinada função. Os dispositivos ou equipamentos escravos não retornam exceções no caso de falhas de comunicação, situação em que estes simplesmente não respondem. Os códigos de exceção são retornados pelos escravos em situações em que a solicitação do mestre (no caso do Driver) foi recebida com sucesso, porém não pôde ser executada por algum motivo, como por exemplo a tentativa de ler ou escrever em um registrador inexistente. Neste caso, o código de exceção retornado indica o tipo de erro ocorrido, ou seja, o motivo pelo qual a requisição do Driver, embora recebida corretamente, não pôde ser executada.

A especificação do protocolo Modbus define nove códigos de exceção. A lista de exceções padrão do protocolo pode ser consultada no tópico **Lista de Exceções Padrão do Protocolo**. Além destes códigos, alguns fabricantes definem códigos adicionais, específicos de seus equipamentos. Tais códigos devem estar documentados no manual do equipamento. Caso não estejam documentados, deve-se consultar o suporte do fabricante.

### Configuração por Strings

- **Dispositivo:** Valor numérico do Id do equipamento (*Slave Id*) seguido de dois pontos. Exemplo: "1:", "2:", "3:", etc.
- **Item:** "LastExceptionCode"

### Configuração Numérica

- **B1:** Endereço do dispositivo escravo (*Slave Id*)
- **B2:** 9999
- **B3:** Não usado, deve ser deixado em zero
- **B4:** Não usado, deve ser deixado em zero

Valores dos Elementos de Bloco retornados:

- **Elemento 1 (índice 0):** Código da exceção retornada pelo equipamento (veja o tópico **Lista de Exceções Padrão do Protocolo**)
- **Elemento 2 (índice 1):** Parâmetro *N2/B2* do Tag de Comunicação que gerou a exceção
- **Elemento 3 (índice 2):** Parâmetro *N3/B3* do Tag de Comunicação que gerou a exceção
- **Elemento 4 (índice 3):** Parâmetro *N4/B4* do Tag de Comunicação que gerou a exceção
- **Elemento 5 (índice 4):** Parâmetro *Size* do Tag de Comunicação que gerou a exceção
- **Elemento 6 (índice 5):** Parâmetro *Dispositivo* do Tag de Comunicação que gerou a exceção
- **Elemento 7 (índice 6):** Parâmetro *Item* do Tag de Comunicação que gerou a exceção

## Utilização do Tag Especial

A utilização mais comum deste Tag durante o *scan* normal dos Tags de funções é através de um evento **OnRead** do Tag de exceção. Neste caso, o script deve antes de tudo rejeitar valores nulos, pois estes indicam o não recebimento de exceções. Em seguida, pode-se tratar a exceção executando os procedimentos adequados, conforme o código recebido. É uma boa prática zerar o registrador de exceção ao sair do script, de forma a indicar que a exceção já foi tratada. Veja o exemplo a seguir, escrito em Elipse Basic (**Elipse SCADA**):

```
// Evento OnRead do Tag TagExc
// Obs.: Para este exemplo, considere TagExc
// com leitura e escrita automática habilitadas

If TagExc == 0
    Return
EndIf

If TagExc == 1
    ... // Trata a exceção 1
ElseIf TagExc == 2
    ... // Trata a exceção 2
Else
    ... // Trata as demais exceções
EndIf

TagExc = 0 // Zera o registrador de exceções
```

A seguir outro exemplo, escrito em VBScript (**Elipse E3** e **Elipse Power**):

```
' Evento OnRead do Tag TagExc
' Obs.: Para este exemplo, considere TagExc
' com leitura e escrita automática habilitadas

Sub TagExc_OnRead()
    If Value = 0 Then
        Exit Sub
    End If

    If Value = 1 Then
        ... ' Trata a exceção 1
    ElseIf Value = 2 Then
        ... ' Trata a exceção 2
    Else
        ... ' Trata as demais exceções
    End If

    Value = 0 ' Zera o registrador de exceções
End Sub
```

Já nas operações de escrita por script, em que seja preciso testar o retorno de exceções logo após o envio do comando, deve-se primeiramente zerar o registrador de exceções. Isto evita que uma eventual exceção provocada pelo comando de escrita seja confundida com outra pré-existente. Executa-se então a operação de escrita e testa-se o valor do Tag Especial, que deve retornar 0 (zero) caso nenhuma exceção tenha sido recebida. Caso retorne um valor diferente de 0 (zero), pode-se então tratar apropriadamente a exceção recebida. Veja o exemplo a seguir, escrito em Elipse Basic (**Elipse SCADA**):

```
// Obs: Para este exemplo, considere TagExc
// com leitura e escrita automática habilitadas
// e TagVal com escrita automática desabilitada

TagExc = 0 // Zera o registrador de exceções

TagVal.WriteEx(10) // Escreve o valor 10

If TagExc <> 0
    ... // Trata a exceção
EndIf
```

A seguir outro exemplo, escrito em VBScript (**Elipse E3** e **Elipse Power**):

```
' Obs: Para este exemplo, considere TagExc
' com leitura e escrita automática habilitadas
' e TagVal com escrita automática desabilitada

' Zera o registrador de exceções
Application.GetObject("Tags.TagExc").Value = 0

' Escreve o valor 10
Application.GetObject("Tags.TagVal").WriteEx(10)

If Application.GetObject("Tags.TagExc").Value <> 0 Then
... ' Trata a exceção
End If
```

#### NOTA

Este Tag Especial retorna, além do código da exceção (retornado no Elemento zero), também os parâmetros do Tag cuja comunicação provocou a exceção. Caso estas informações não sejam necessárias, pode-se perfeitamente ler o mesmo registro através de um Tag simples (Tag PLC no **Elipse SCADA**), sem necessidade de usar Tags Bloco. Neste caso, os procedimentos recomendados permanecem os mesmos.

## Leitura de Memória de Massa

Este Driver permite definir, nas operações, funções especiais de leitura para a coleta de memória de massa de dispositivos escravos. Tais funções não existem no protocolo, e implicam no uso de algoritmos específicos de leitura de eventos dos equipamentos, que podem eventualmente ler ou escrever em diversos registros, utilizando-se de uma ou mais funções do protocolo.

### Leitura por Callbacks

A partir da versão 2.08, este Driver passou a implementar a leitura por *callbacks*, recurso disponível no **E3** (a partir da versão 3.0) e no **Elipse Power**, que otimiza o desempenho de leituras de memória de massa. Com este recurso disponível, a aplicação delega ao Driver a varredura dos Tags de leitura de eventos de memória de massa. Em outras palavras, a aplicação não precisa mais interrogar constantemente o Driver a cada período de varredura. Ao invés disto, o Driver realiza a verificação de novos eventos no equipamento e coleta eventos sempre que estiverem disponíveis, tomando a iniciativa de enviá-los à aplicação.

### Funções Especiais para Leitura de Memória de Massa

Na atual versão do Driver, as seguintes funções de leitura de sequências de eventos (SOE) são suportadas:

- **GE SOE:** Realiza a coleta de eventos de CLPs GE PAC RX7. Para mais informações, consulte o tópico **Leitura de Buffer de Eventos em Controladores GE PAC RX7**
- **SP SOE:** Coleta eventos de relés Schneider Electric SEPAM séries 20, 40 e 80. Para mais informações, consulte o tópico **Leitura de Eventos de Relés Schneider Electric SEPAM 20, 40 e 80**
- **GenSOE:** Esta função usa o algoritmo genérico criado pela **Elipse Software**, o **Elipse Modbus SOE**, que pode ser usado pela maioria dos controladores programáveis. Exige a criação de procedimento análogo na programação do CLP (*ladder*). Para mais informações, consulte o tópico **Algoritmo de Leitura de SOE Genérico da Elipse Software**
- **65 03:** Função especial para a leitura de eventos de memória de massa de medidores ABB MGE 144. Para mais informações, consulte o tópico **Leitura de Registros da Memória de Massa de Medidores ABB MGE 144**

## Leitura de Buffer de Eventos em Controladores GE PAC RX7

O *buffer* de eventos pode ser lido através de três tipos de Tags: **Tags reportados por eventos**, **Tags reportados por eventos por ponto** e **Tags de tempo real**.

### Tags Reportados por Eventos

Os Tags reportados por eventos retornam, a cada operação de leitura, todos os eventos acumulados no *buffer* interno do Driver, podendo ser **configurados por Strings** ou **numericamente**.

### Configuração por Strings

- **Dispositivo:** "<Slave Id>:"
- **Item:** "gesoe0.<endereço base da pilha de eventos>"

### Configuração Numérica

Para a leitura do *buffer* de eventos do GE PAC RX7 usando a configuração numérica, deve ser definida, na janela de configuração do Driver, uma operação que use como função de leitura a **função especial GE SOE**. O tipo de dados deve ser definido como **GE\_events**.

- **B1:** *Slave ID*
- **B2:** Código da operação definida com a função **GE SOE**
- **B3:** 0 (zero)
- **B4:** Endereço base da pilha de eventos no CLP

A cada *scan* neste Tag, o Driver verifica se existem eventos no *buffer* do controlador. Se houver eventos, o Driver inicia uma *thread* de leitura de eventos, que é executada em segundo plano, não bloqueando a varredura dos demais Tags. Após o término da leitura do *buffer* pelo Driver, este Tag reportado por eventos retorna o conjunto de eventos lidos na varredura.

Os eventos retornados geram uma sucessão de eventos **OnRead** neste Tag. Para cada evento lido, o E3 atualiza os campos do Tag (valores de Elementos e *timestamp*) com os valores de um determinado evento, e executa uma vez o evento **OnRead**. O script do evento executado deve ser definido pelo usuário, sendo geralmente usado para inserir os dados do Tag no Histórico.

Cada evento é representado por um Bloco de dois Elementos, com o campo **Timestamp** lido do equipamento. Os campos do respectivo Tag Bloco de leitura são mostrados na tabela a seguir.

**Campos do Tag Bloco**

OFFSET	SIGNIFICADO	TIPO DE DADOS	FAIXA DE VALORES
<b>0</b>	Identificação do ponto	Byte	Entre 0 e 15
<b>1</b>	Status do ponto	Byte	Entre 0 e 1

Para mais informações sobre Tags reportados por evento, consulte o tópico específico no *Manual do Usuário do E3*.

**IMPORTANTE**

Ao ler eventos de memória de massa em Tags reportados a eventos no **E3**, desabilite a banda morta do Tag (propriedade **EnableDeadBand** configurada para Falso) e também do objeto Histórico associado (propriedade **DeadBand** igual a zero), para evitar a perda de eventos com valores próximos. Também é importante desabilitar o histórico por varredura (no **E3**, propriedade **ScanTime** igual a zero). Com isto, garante-se que novos eventos só são armazenados através do método **WriteRecord**, executado no evento **OnRead** do Tag, evitando a duplicação de eventos.

**Tags Reportados por Eventos por Ponto**

A partir da versão 2.5 do Driver é possível utilizar um novo Tag para o *download* de eventos de um ponto específico.

Este Tag funciona de forma idêntica ao anterior, exceto pelo fato de retornar apenas os eventos de um ponto específico.

Ao contrário do anterior, o valor retornado possui apenas um Elemento com o valor do status do ponto, de forma que pode-se utilizar apenas um Tag. Este Tag deve ser configurado da seguinte forma:

**Configuração por Strings**

- **Dispositivo:** "<Slave Id>:"
- **Item:** "gesoe<200 + Índice do ponto>.<endereço base da pilha de eventos>"

**Configuração Numérica**

- **N1:** *Slave ID*
- **N2:** Código da operação definida com a função **GE SOE**
- **N3:** 200 + Índice do ponto (por exemplo, para o ponto 2, configure *N3* como 202)
- **N4:** Endereço base da pilha de eventos no CLP

Para mais informações sobre Tags reportados por evento, consulte o tópico específico no *Manual do Usuário do E3*.

**Tags de Tempo Real**

Estes Tags retornam o evento mais recente já lido para um ponto específico. Estes eventos são armazenados na memória interna do Driver a cada leitura de eventos do CLP, com seus respectivos *timestamps* lidos do equipamento. Este Tag utiliza os seguintes parâmetros:

**Configuração por Strings**

- **Dispositivo:** "<Slave Id>:"
- **Item:** "gesoe<100 + Índice do Ponto>.<endereço base da pilha de eventos>"

**Configuração Numérica**

- **N1:** *Slave ID*
- **N2:** Código da operação
- **N3:** 100 + Índice do ponto

- **N4:** Endereço base da pilha de eventos no CLP
- **Valor:** Status do ponto

## Leitura de Eventos de Relés Schneider Electric SEPAM 20, 40 e 80

Para a leitura de relés SEPAM, o modelo de *offset* de endereçamento deve ser configurado como **Data is addressed from 0**, na aba **Modbus**. A leitura de eventos é realizada com a utilização de dois tipos básicos de Tags:

- **Tag de coleta de todos os eventos da tabela (obrigatório):** Realiza a coleta de todos os eventos da tabela de uma determinada zona na memória do equipamento. Este Tag, além de retornar todos os eventos lidos à aplicação reportados a evento, ainda acumula os eventos lidos no *buffer* interno do Driver, para serem retirados através de leituras nos Tags de leitura de evento único, descritos a seguir
- **Tag de leitura de evento único (opcional):** Retorna eventos recebidos do relé especificado, com determinado endereço, tipo e zona. Este Tag não realiza a leitura direta do equipamento, mas sim retorna os eventos do *buffer* interno do Driver, alimentado durante a leitura do Tag de coleta de todos os eventos, descrito anteriormente, ou seja, para que seja possível ler eventos com este tipo de Tag, um Tag do tipo **Coleta de todos os eventos** deve já estar previamente ativo, com sua varredura (*scan*) habilitada. Este Tag é útil quando o usuário precisa obter eventos de um tipo e de uma fonte específicas (relé, zona, endereço e tipo). Um exemplo de utilização é a associação a objetos de Tela, mostrando o status de determinado endereço de evento. Embora este Tag retorne as mesmas informações já retornadas pelo Tag anterior, seu uso poupa o usuário de ter que implementar filtros, cláusulas **Select Case** em VBScript ou algum outro método para separar os diversos tipos de eventos retornados pelo **Tag de coleta de todos os eventos por script** na aplicação

A aplicação deve necessariamente implementar um Tag de coleta de todos os eventos para cada tabela ou zona de eventos a ser coletada em cada relé, pois é durante a leitura deste Tag que os eventos de fato são coletados do equipamento. A seguir é descrita a configuração destes dois Tags.

### Tag de Coleta de Todos os Eventos da Tabela (Zona de Eventos)

Este Tag é reportado a eventos. Sua aplicação típica é a inserção de eventos em um objeto Histórico associado, através do método **WriteRecord** do Histórico, executado no evento **OnRead** do Tag. A cada leitura, ou seja, a cada período de varredura do Tag, o Driver pode coletar até quatro novos eventos do equipamento. Este é o número máximo de eventos que cada zona de eventos do relé dispõe a cada requisição de leitura.

Como é durante a leitura deste Tag que os eventos são de fato coletados do equipamento, mesmo que seus dados não sejam utilizados diretamente, ou seja, mesmo **que não seja preciso armazenar todos os eventos coletados em Histórico, sua implementação é obrigatória** para que os Tags de evento único possam retornar dados. O Tag de coleta de todos os eventos deve ser configurado como um Tag Bloco com três Elementos, da seguinte forma:

### Configuração por Strings

- **Dispositivo:** "<Slave Id>:"
- **Item:** "spsoe<Zona ou Tabela de Eventos (1 ou 2)>"

Exemplo: Para a leitura da Zona 1 do Escravo 1, **Dispositivo** é igual a "1:" e **Item** é igual a "spsoe1". Alternativamente, **Dispositivo** pode ser igual a "" (**String** vazia) e **Item** igual a "1:spsoe1" (veja o tópico **Configuração por Strings**).

### Configuração Numérica

Para usar a configuração numérica deve ser definida, na aba **Operations**, uma operação que use como função de leitura a **função especial SP SOE**. O tipo de dados é definido automaticamente como **SP\_events**, assim que a função de leitura **SP SOE** é selecionada.

- **B1:** 1000 + Endereço do escravo (relé) na rede (entre 1 e 247)
- **B2:** **Código da operação** configurada com a **função de leitura especial SP SOE**



- **B3:** 0 (zero)
- **B4:** Zona ou tabela de eventos (1 ou 2)

A tabela a seguir descreve o significado dos três Elementos de Bloco, que têm seus valores retornados como reportados a eventos.

#### Significados dos Elementos de Bloco (tipo de dados SP\_events)

OFFSET	SIGNIFICADO	TIPO DE DADOS	FAIXA DE VALORES
<b>0</b>	Tipo de evento	<b>Word</b>	De 0 a 65535 (800H para <b>Remote Annunciation, Internal Data e Logic Input</b> )
<b>1</b>	Endereço do evento	<b>Word</b>	Referencia endereços de bits de 1000H a 105FH
<b>2</b>	Borda de subida ou de descida	<b>Word</b>	<ul style="list-style-type: none"> <li>• <b>00:</b> Borda de descida</li> <li>• <b>01:</b> Borda de subida</li> </ul>

Para mais informações sobre Tags reportados a eventos, veja o tópico **Tags Reportados a Eventos** do *Manual do Usuário do E3*.

#### IMPORTANTE

Ao ler eventos de memória de massa em Tags reportados a eventos no **E3**, desabilite a banda morta do Tag (propriedade **EnableDeadBand** configurada para Falso) e também no objeto Histórico associado (propriedade **DeadBand** igual a zero), para evitar a perda de eventos com valores próximos. Também é importante desabilitar o histórico por varredura (no **E3**, propriedade **ScanTime** igual a zero). Com isto, garante-se que novos eventos só são armazenados através do método **WriteRecord**, executado no evento **OnRead** do Tag, evitando a duplicação de eventos.

### Tag de Leitura de Evento Único

Este Tag é também reportado a eventos, sendo possível também usar seu evento **OnRead** para armazenamento em Histórico. Note que nada impede, entretanto, que ele seja tratado como um Tag normal (Tag de tempo real), caso só interesse o seu valor mais recente. Como este Driver apenas lê um *buffer* interno, sugere-se definir um tempo de varredura bem baixo, até menor que o do outro tipo de Tag. O consumo de CPU de cada varredura pode ser considerado desprezível. Sugere-se a metade do período de varredura do Tag de coleta de todos os eventos.

Como já comentado, este Tag é utilizado para obter o status de determinado endereço de evento, sem precisar separar ou realizar filtros nos eventos que chegam pelo Tag anterior, por script ou outro meio. Uma aplicação típica seria a associação a objetos de Tela.

O Tag de leitura de evento único, como já mencionado, não faz a leitura de eventos do equipamento, mas sim de um *buffer* interno do Driver, previamente preenchido durante a leitura do Tag de coleta de todos os eventos. O Tag retorna apenas um Elemento, reportado a eventos, podendo ser configurado como Tag simples (não precisa ser um Tag Bloco). O Driver aceita no máximo oito eventos acumulados por ponto de evento, ou seja, para cada combinação de relé, zona, tipo e endereço do evento, em seu *buffer* interno. Se ocorrer *overflow*, ou seja, se mais de oito eventos de um mesmo ponto forem retornados sem que nenhum Tag de evento único os colete, o Driver passa a descartar os eventos mais antigos. A configuração adequada do tempo de varredura pode evitar a perda de eventos.

## DICA

Recomenda-se configurar a varredura (*scan*) dos Tags de evento único com um valor equivalente à metade do configurado para o Tag de coleta de todos os eventos associado, evitando-se assim a perda de eventos por *overflow* do *buffer* interno do Driver.

Deve ser configurado com os seguintes parâmetros:

### Configuração por Strings

- **Dispositivo:** "<Slave Id>:"
- **Item:** "ptspsoe<Tipo do evento (0800H por padrão)>.<Event bit address + Events zone offset\* (ver tabela a seguir)>"

Exemplo: Para a leitura de eventos do tipo **800H** no endereço 1 da zona 2, **Dispositivo** deve ser igual a "1:" e **Item** deve ser igual a "ptspsoe&h800.&h8001". Alternativamente, **Dispositivo** pode ser igual a "" e **Item** igual a "1:ptspsoe&h800.&h8001" (veja o tópico **Configuração por Strings**).

### Configuração Numérica

- **N1:** Endereço do escravo (relé) na rede (entre 1 e 247)
- **N2: Código da operação** configurada com a **função especial SP SOE**
- **N3:** Tipo do evento (0800H por padrão, segundo documentação do fabricante)
- **N4:** Endereço do evento (*Event bit address*) + *Events Zone Offset*, conforme descrito na tabela a seguir

#### Opções para o endereço do evento (Events Zone Offset)

ZONA DE EVENTOS	EVENTS ZONE OFFSET
1	0
2	8000H (8000 em hexadecimal)

Exemplos:

- **Endereço do Evento:** 102FH, Zona de Evento = 1 ® N4 = 102FH + 0 = 102FH
- **Endereço do Evento:** 102FH, Zona de Evento = 2 ® N4 = 102FH + 8000H = 902FH

## NOTA

Para representar valores em formato hexadecimal no **Elipse E3** e no **Elipse Power**, deve-se usar o prefixo "&H" (por exemplo, &H10 = 16). No **Elipse SCADA**, use o sufixo "h" (por exemplo, 10h = 16). Neste Manual, entretanto, é usado o sufixo "H" para denotar valores no formato hexadecimal.

- **Valor:** Retorna a borda de subida ou descida, conforme a tabela a seguir

#### Opções disponíveis para Valor

VALOR	SIGNIFICADO
00	Borda de descida
01	Borda de subida

- **Timestamp:** A propriedade **Timestamp** representa a data e a hora em que o evento foi de fato lido do relé, durante a leitura do Tag de leitura de todos os eventos descrito anteriormente

Para maiores informações sobre os eventos do relé, seus significados e endereçamento, consulte a documentação do fabricante. Para mais informações sobre Tags reportados a eventos, veja o tópico **Tags Reportados a Eventos** do *Manual do Usuário do E3*.

# Algoritmo de Leitura de SOE Genérico da Elipse Software

O protocolo Modbus não define um método padrão de leitura de eventos do equipamento. Por este motivo, é comum fabricantes criarem seus próprios algoritmos de leitura de eventos em equipamentos com suporte ao protocolo Modbus.

O algoritmo de Sequenciamento de Eventos (SOE, *Sequencing of Events*) genérico deste Driver (**Elipse Modbus SOE**) foi desenvolvido pela **Elipse Software** com o objetivo de prover uma alternativa padrão para a leitura de eventos de controladores programáveis que não contem com este recurso de forma nativa, desde que estes atendam a alguns requisitos básicos de espaço de memória e recursos de programação, permitindo criar as tabelas e registros de controle descritos a seguir.

Este algoritmo permite armazenar e ler eventos de praticamente qualquer controlador programável, de forma otimizada, valendo-se de recursos já implementados e validados no Driver Modbus.

A leitura de eventos no Driver Modbus segue o procedimento padrão, definido pela **Elipse Software**, lendo eventos de **tabelas montadas na memória do CLP** ou de dispositivos escravos pelo seu software residente ou aplicativo (*ladder*).

Para utilizar este algoritmo é preciso definir Tags com a **função especial Gen SOE** do Driver, o que pode ser realizado tanto através da antiga **configuração numérica** (parâmetros *N* e *B*) como pelo novo método de **configuração por Strings** (campos **Dispositivo** e **Item**). A configuração dos Tags é descrita mais adiante no tópico **Procedimento de Aquisição na Aplicação**.

Durante o processo de leitura de eventos, a **função especial Gen SOE** usa sempre a função Modbus **03** (*Read Holding Register*) para leitura de registros do equipamento. Já para a escrita na atualização dos registros de controle, é usada por padrão a função **16** (*Write Multiple Registers*). Na configuração numérica é possível selecionar a função de escrita **06** (*Write Single Registers*), para o caso raro de equipamentos que não suportem a função **16** (o contrário é mais comum), através do campo **Write** da operação, na **aba Operations**.

O **software residente do CLP** (*ladder* ou equivalente) deve manter atualizados os registradores de controle que fornecem informações ao Driver, como o número de eventos disponíveis para a leitura e o endereço do último registro a ler.

O equipamento pode manter mais de uma **tabela de eventos**, em diferentes endereços de memória, contendo tipos de dados diferentes. Cada tabela deve ser precedida pelos seus respectivos **registradores de controle**, em endereços adjacentes. A tabela em si é constituída por um **buffer circular** em endereços contíguos, acumulando os eventos ou dados a serem coletados pelo Driver a cada procedimento de coleta (*download* de eventos).

O usuário pode definir formatos distintos de dados (eventos) para cada uma das tabelas definidas, os quais são geralmente definidos como uma **estrutura de dados**, podendo conter o campo de estampa de tempo do evento (*timestamp*). Os eventos podem também ser definidos usando um **tipo de dados nativo** do Driver. Neste caso, não é possível definir um campo **Timestamp** no CLP (o *timestamp* vem com a data da leitura), e o evento tem somente um campo, podendo ser representado por um Tag simples (Tag PLC no **Elipse SCADA**).

## NOTA

O algoritmo de SOE sempre usa a função **03** (*Read Holding Registers*) do protocolo Modbus para a leitura de registros do equipamento. Para a escrita de registros é usada por padrão a função **16** (*Write Multiple Registers*). Também é possível selecionar a função **06** (*Write Single Register*) na **configuração numérica** apenas, através do campo **Write** da respectiva operação, na **aba Operations**.

Os tópicos a seguir descrevem de forma detalhada o algoritmo, sua implementação no software do CLP (*ladder*) e como realizar sua leitura usando os Tags do Driver:

- **Tabela de Eventos**
- **Procedimento de Aquisição no CLP**

- **Procedimento de Aquisição na Aplicação**

## Tabela de Eventos

Como já mencionado no tópico **Algoritmo de Leitura de SOE Genérico da Elipse Software**, cada tabela de eventos mantém os eventos em um *buffer* circular. O *buffer* circular de cada tabela é definido pelo seu endereço inicial, ou endereço base, contíguo aos registradores de controle, e por seu número máximo de registros, que define seu limite final. A tabela a seguir ilustra a disposição dos eventos dentro do *buffer* circular de uma tabela.

**Disposição de eventos do buffer circular**

EVENTO	TIMESTAMP	ELEMENTO1	ELEMENTO2	ELEMENTO3	...	ELEMENTON
1						
2						
3						
4						
5						
...						
N						

Cada linha da tabela anterior representa um evento armazenado, normalmente representado por uma estrutura, ou por **Tipos de Dados Definidos pelo Usuário**.

Note que, no exemplo da tabela anterior, o primeiro elemento da estrutura dos eventos é o *timestamp*. Este campo, cuja presença não é obrigatória, e que não necessariamente precisa aparecer na primeira posição, define a propriedade **Timestamp** do Tag, não sendo retornado em seus Elementos (para mais informações, veja o tópico **Tipos de Dados Definidos pelo Usuário**).

É possível também definir eventos com **tipos de dados nativos do Driver**, e neste caso tem-se apenas um Elemento de dado em cada evento, sem *timestamp*.

Os eventos devem ser inseridos no *buffer* circular em ordem crescente, retornando ao endereço base após atingir o limite superior do *buffer* circular. Os seguintes registros de controle devem ser definidos para cada tabela:

- **Status da tabela:** Deve ser mantido pelo CLP, indicando o número de eventos disponíveis para a leitura no *buffer* circular. Deve ser atualizado pelo dispositivo sempre que novos eventos forem adicionados ao *buffer* circular, ou após a conclusão da coleta de eventos pela aplicação, o que pode ser detectado pelo **Status da aquisição**
- **Ponteiro de gravação:** Este valor indica o índice, começando em 0 (zero), da posição onde o dispositivo deve inserir o próximo evento. Deve ser incrementado pelo dispositivo a cada nova inserção de eventos no *buffer* circular, voltando ao endereço base após alcançar o limite superior do *buffer*. Note que este valor não deve ser fornecido em unidades de registradores Modbus, mas sim em posições de eventos, devendo ser incrementado em uma unidade a cada novo evento inserido, independente do número de registradores Modbus ocupados para cada evento no *buffer* circular. Com isto, o valor máximo permitido para este ponteiro pode ser dado pela fórmula **MaxWritePtr = (Tamanho do buffer circular / Tamanho da estrutura de evento) - 1**
- **Status da aquisição:** Indica o número de registros já lidos pelo Driver a cada leitura individual de eventos. Após cada leitura, o Driver escreve neste registro o número de registros que conseguiu ler. O aplicativo residente no escravo (*ladder*) deve então imediatamente subtrair o valor escrito pelo Driver do **Status da tabela** e então zerar o **Status da aquisição**
- **Reservado:** Este registrador atualmente não é utilizado. Pode ser usado em futuras versões do Driver, podendo ser deixado em 0 (zero) na versão atual

Conforme já mencionado, o endereço base do *buffer* circular, ou seja, o endereço onde inicia a tabela de eventos, deve ser contíguo aos registradores de controle.

Já os registradores de controle devem estar dispostos também em endereços contíguos, na ordem apresentada anteriormente, permitindo sua leitura em uma única operação, ou seja, supondo-se que o endereço base dos registradores de controle para uma determinada tabela seja 100, tem-se os seguintes endereços para os demais registradores:

#### Endereços de registradores

REGISTRADOR	ENDEREÇO
Status da Tabela	100
Ponteiro de Gravação	101
Status da Aquisição	102
Reservado	103
Endereço Base do Buffer Circular	104

No tópico **Procedimento de Aquisição no CLP** é descrito passo a passo o procedimento ou algoritmo de aquisição sob o ponto de vista do equipamento escravo (CLP). No tópico seguinte, **Procedimento de Aquisição na Aplicação**, discute-se como configurar a aplicação para a aquisição dos eventos da tabela.

## Procedimento de Aquisição no CLP

Neste tópico discute-se o algoritmo de coleta de eventos sob o ponto de vista do CLP ou dispositivo escravo. O objetivo é deixar claro ao desenvolvedor o que é preciso ser implementado no software residente do CLP (*ladder*).

O dispositivo deve iniciar a inserção dos eventos no sentido crescente, a partir do endereço base da tabela, ou seja, do *buffer* circular. A cada novo evento inserido, o Ponteiro de Gravação deve ser incrementado, passando a apontar para a próxima posição disponível do *buffer*.

O Driver realiza a leitura do evento mais antigo para o mais recente. O endereço do início da leitura é calculado pelo Driver através do valor do Ponteiro de Gravação e do Status da Tabela.

Se o número de eventos disponíveis for maior que o **máximo permitido em um único frame de comunicação** do protocolo, o Driver realiza múltiplas leituras em bloco, atualizando após a conclusão do processo o valor do Status da Aquisição com o número total de eventos lidos.

### NOTA

Caso o equipamento não respeite o limite padrão de 253 bytes para o PDU, é preciso configurar a opção **Customize Max. PDU Size**, na aba **Modbus**, de acordo com os limites suportados, que devem estar descritos na documentação do fabricante.

Ao detectar um valor não nulo escrito pelo Driver no Status da Aquisição, o aplicativo do dispositivo ou CLP deve imediatamente subtrair o valor do Status da Aquisição do valor do Status da Tabela e em seguida zerar o Status da Aquisição. Com o Status da Aquisição novamente zerado, o Driver pode iniciar nova aquisição a qualquer momento.

O CLP pode inserir novos eventos na tabela durante o processo de aquisição pelo CLP, desde que não ocorra *overflow* do *buffer* circular, ou seja, desde que o ponteiro de escrita não ultrapasse o de leitura, incrementando o Status da Tabela.

O procedimento de coleta ou *download* de eventos é concluído quando o Status da Tabela for zerado. Os eventos coletados são então disponibilizados à aplicação por meio de Tags reportados a evento, procedimento visto no tópico a seguir.

A seguir é apresentado um pequeno fluxograma, em formato de Diagrama de Atividade UML, com sugestão de implementação para a lógica do CLP. Note que são possíveis algumas variações, como por exemplo descartar o evento mais antigo em caso de *overflow*, que podem ser avaliadas pelo desenvolvedor, dependendo do contexto.



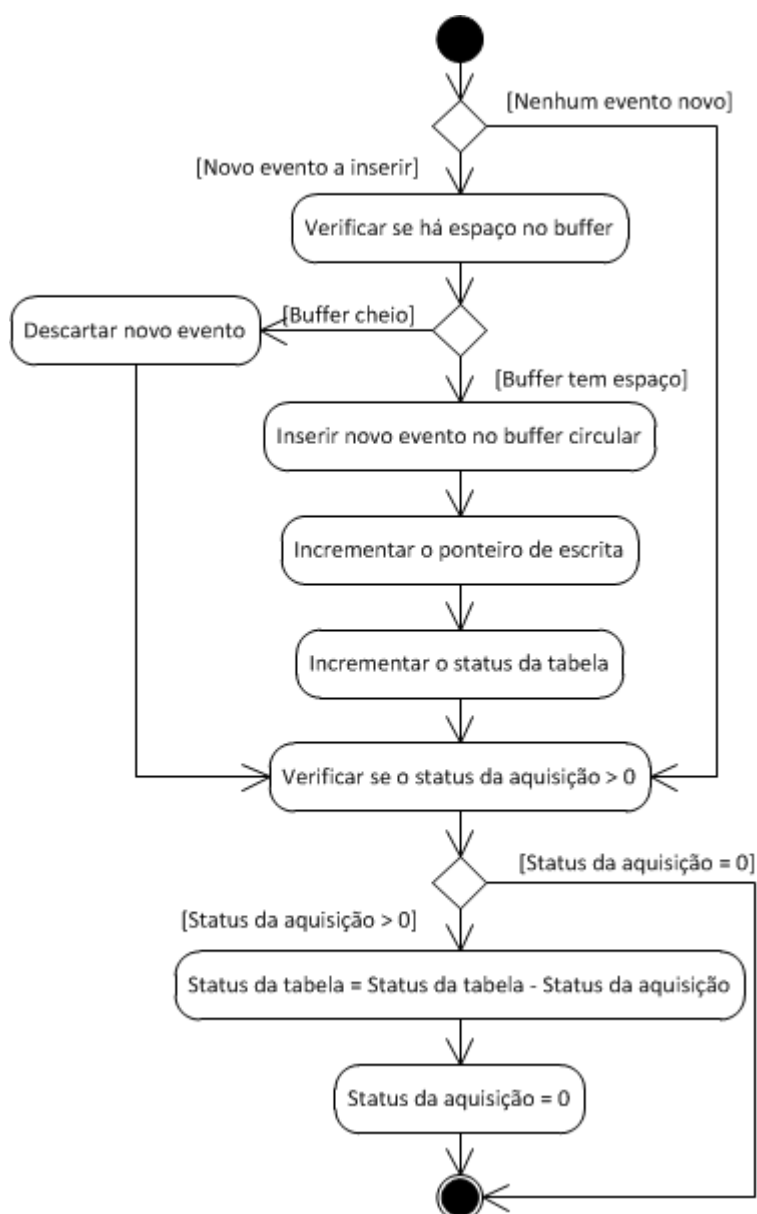


Diagrama de Atividade UML (CLP)

## Estampa de Tempo

Conforme já mencionado, cada evento é composto por uma estrutura contendo um ou mais elementos de dados (geralmente, porém não necessariamente, representados por **Tipos de Dados Definidos pelo Usuário**).

Se forem usadas estruturas (tipos de dados definidos pelo usuário), é possível associar a cada evento uma estampa de tempo (*timestamp*) fornecida pelo CLP. Neste caso, o valor do campo **Timestamp** deve ser fornecido em um campo da estrutura, na memória do CLP, na ordem em que é declarado no arquivo de configuração, e seu valor não é mostrado em nenhum Elemento de Bloco, sendo retornado apenas na propriedade **Timestamp** do Tag associado.

Conforme explicado no tópico **Tipos de Dados Definidos pelo Usuário**, qualquer tipo de dados de data e hora suportado pelo Driver pode ser usado. O tipo de dados **GenTime**, entretanto, foi criado especialmente para uso com o **Eclipse Modbus SOE**, devido à facilidade de definição no software residente (*ladder*) do CLP.

Caso não seja necessária uma precisão de milissegundos, outra opção a considerar é o tipo de dados **UTC32** do Driver, representado como um inteiro de apenas 32 bits (quatro bytes) com os segundos desde 1/1/1970, sem a representação dos milissegundos, considerados como 0 (zero).

No tópico seguinte, **Procedimento de Aquisição na Aplicação**, descreve-se como configurar o supervisor para a coleta dos eventos acumulados no CLP ou dispositivo escravo programável.

## Procedimento de Aquisição na Aplicação

Neste tópico é detalhada a configuração da aplicação do supervisor para a aquisição dos eventos acumulados no CLP ou dispositivo escravo programável.

A leitura dos eventos no supervisor é realizada por meio de Tags que usam a função especial de leitura **Gen SOE**. O tipo de dados do Tag define a estrutura dos eventos armazenados na tabela do equipamento. Se for definido um **tipo de dados nativo do Driver** (*built-in type*), cada evento tem apenas um elemento deste tipo de dados, sem *timestamp* fornecido pelo CLP (o *timestamp* representa o momento da coleta dos eventos). Por outro lado, se forem usados **tipos de dados definidos pelo usuário**, é possível definir estruturas para os eventos, incluindo *timestamps*, como visto mais adiante neste tópico.

A seguir é descrita a configuração dos Tags tanto na nova metodologia de configuração por **Strings** (campos **Dispositivo** e **Item**), como na antiga configuração numérica do **Eclipse SCADA** (parâmetros *N* e *B*).

### Configuração por Strings

- **Dispositivo:** "<Slave Id>:"
- **Item:** "elsoe<N>.<end. inicial>[.<tipo><tam. do tipo>]][.<byte order>][bit]"

Onde:

- **N:** Tamanho da tabela no dispositivo, em número máximo de eventos comportados
- **end. inicial:** Endereço do primeiro registro de controle, usando o valor definido na tabela exemplo do tópico **Tabela de Eventos**
- **tipo:** Tipo de dados nativo ou do usuário usado para cada evento (veja o tópico **Configuração por Strings**)
- **tam. do tipo:** Usado apenas para tipos de dados de tamanho variável (veja o tópico **Configuração por Strings**)
- **byte order:** Ordenamento de bytes. Deve ser omitido para equipamentos que seguem plenamente o padrão do protocolo (veja o item **Byte Order** no tópico **Configuração por Strings** para mais informações). Quando são usadas estruturas, afeta apenas seus elementos individuais (veja o tópico **Tipos de Dados Definidos pelo Usuário**)
- **bit:** Mascaramento de bits. Em geral pode ser omitido, dificilmente seria usado aqui (veja o campo **Bit** no tópico **Configuração por Strings**)

Exemplo:

- **Dispositivo:** "1:"
- **Item:** "elsoe150.&h101.TYPE3"

O tipo **TYPE3** está definido da seguinte forma no arquivo padrão de exemplo do Driver (veja o tópico **Tipos de Dados Definidos pelo Usuário**):

```
// This type has an UTC32-type timestamp
// and a few named elements
struct TYPE3
{
    DefaultAddress = 0x101;
    timestamp = UTC32;
    float Va;
    float Vb;
    float Vc;
    float Ia;
    float Ib;
    float Ic;
}
```

Trata-se, portanto, de um tipo de dados **Estrutura** com seis campos de dados e *timestamp*. Daí deduz-se que o Tag deve ser um Bloco com seis Elementos para representar a estrutura.

Note que, conforme já explicado, o valor do *timestamp*, embora ocupe registros no CLP, não necessita de Elementos de Tag Bloco, pois seu valor é retornado no campo *timestamp* do Tag.

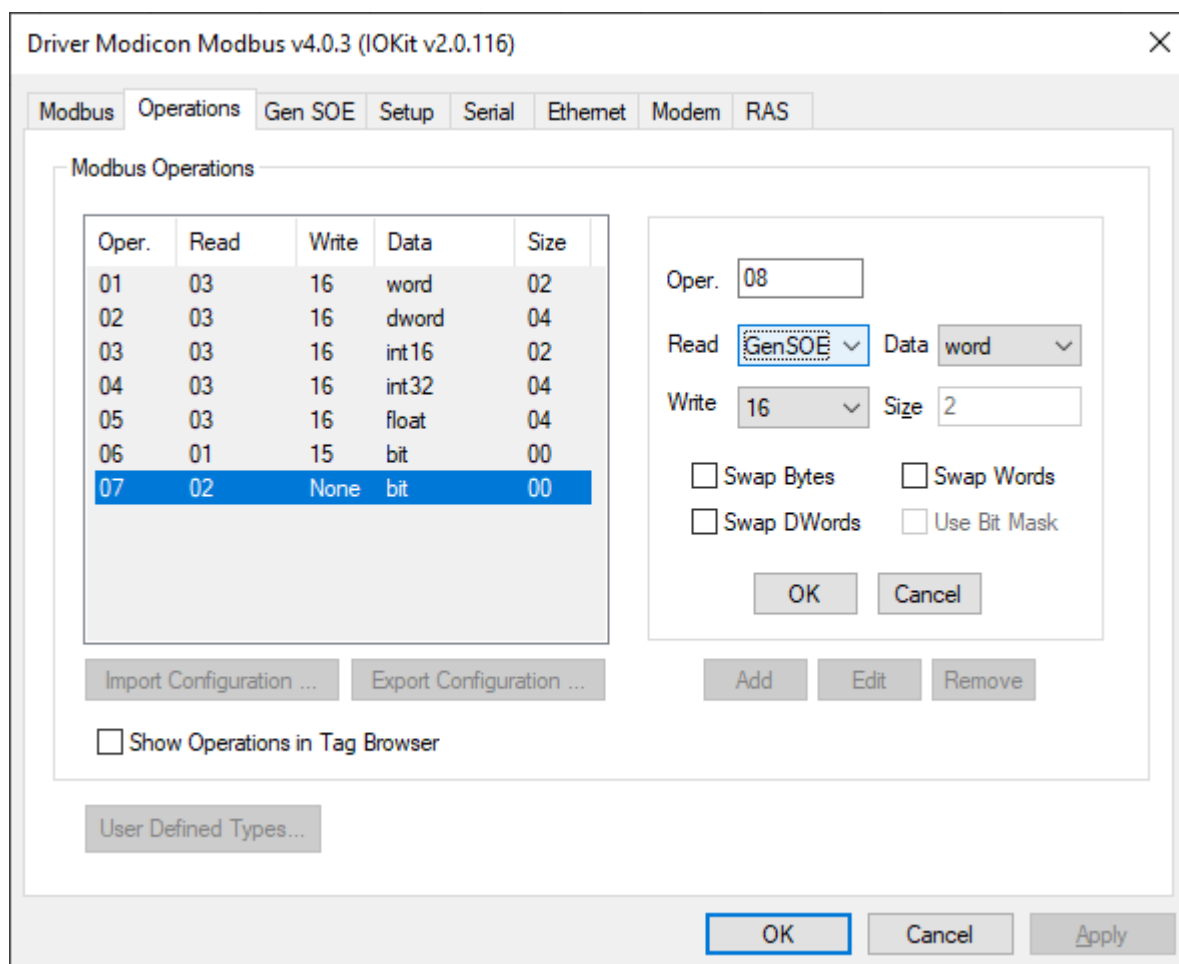
#### NOTA

O parâmetro *N* informa o tamanho da tabela em número máximo de eventos suportados, e não em registros Modbus. Em conjunto com o parâmetro *Endereço Inicial*, informa indiretamente o endereço final ou limite superior da tabela. O tamanho da área de dados da tabela, portanto, em número de registros Modbus, é o produto de *N* pelo tamanho de cada evento em número de registros Modbus, ou seja, em **Words** de 16 bits.

### Configuração Numérica (Parâmetros N e B)

Para configurar Tags de leitura de Eclipse SOE usando a configuração numérica, é necessário configurar uma operação na **aba Operations**, usando a **função especial GenSOE**.

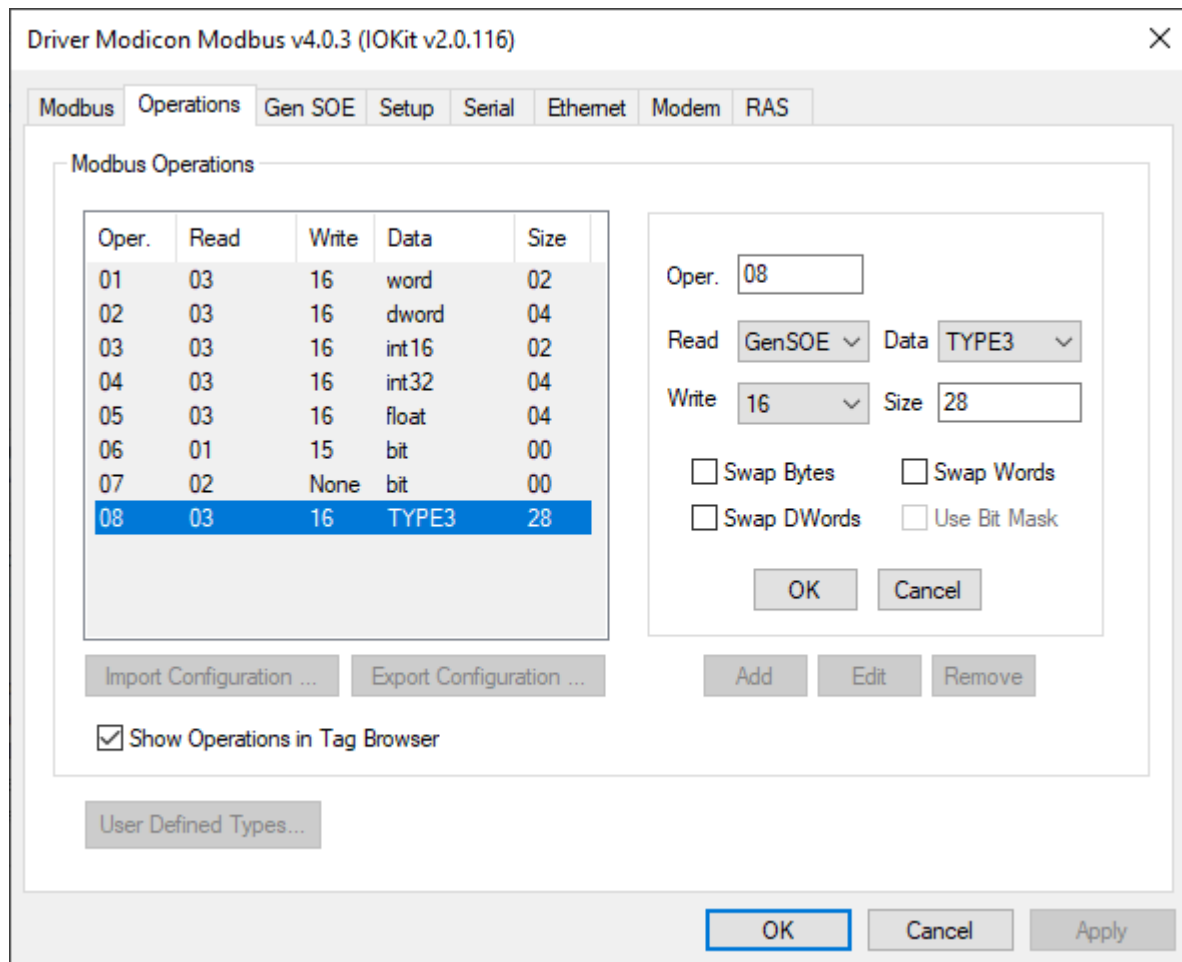
A figura a seguir mostra um exemplo de adição de operação utilizando a função especial **GenSOE** com o tipo de dados **Word**.



#### Função especial GenSOE

Note que foi selecionada a função **16** (*Write Multiple Registers*) como função de escrita, que é a função mais comum. No entanto, recomenda-se o uso da função **06** (*Write Single Register*) sempre que ela for suportada pelo equipamento.

A figura a seguir mostra a mesma operação com o tipo de dados definido pelo usuário **TYPE3** (veja o tópico **Tipos de Dados Definidos pelo Usuário**), que é um tipo de dados definido no arquivo de configuração exemplo, disponível com o Driver e que é usado como exemplo neste tópico.



**Configuração com a função GenSOE e um tipo de dados definido pelo usuário**

O tipo de dados **TYPE3** está definido da seguinte forma no arquivo de exemplo do Driver:


```
// This type has an UTC32-type timestamp
// and a few named elements
struct TYPE3
{
    DefaultAddress = 0x101;
    timestamp = UTC32;
    float Va;
    float Vb;
    float Vc;
    float Ia;
    float Ib;
    float Ic;
}
```

Trata-se, portanto, de um tipo de dados **Estrutura** com seis campos de dados e *timestamp*, e com endereço padrão (parâmetro *B4* do Tag) igual a "101H" (257 em decimal). Para sua leitura, portanto, é preciso definir um Tag Bloco de seis Elementos com a seguinte configuração:

- **B1:** Endereço do dispositivo escravo (CLP) na rede (*Slave Id*)
- **B2:** 8 (operação definida anteriormente com a função especial **GenSOE**)
- **B3:** N (tamanho da tabela no dispositivo, em número máximo de eventos comportados)
- **B4:** 100 (endereço do primeiro registro de controle, usando o valor definido na tabela exemplo do tópico **Tabela de Eventos**)
- **Size:** 6

**NOTA**

O parâmetro *B3* informa o tamanho da tabela em número máximo de eventos suportados, e não em registros Modbus. Em conjunto com o parâmetro *B4*, informa indiretamente o endereço final ou limite superior da tabela. O tamanho da área de dados da tabela, portanto, em número de registros Modbus, é o produto de *B3* pelo tamanho de cada evento em número de registros Modbus, ou seja, em **Words** de 16 bits.

Note que, caso o Tag Browser do **E3** seja usado para inserir o Tag na aplicação, como explicado no tópico **Tipos de Dados Definidos pelo Usuário**, Elementos do Tag já são nomeados conforme o nome dado aos elementos da estrutura em sua declaração. O Tag Browser pode ser aberto clicando-se em  na aba **Design** do Driver.

**Utilização**

Uma vez tendo definido o Tag (ou Tags) apropriado, habilite sua varredura e deixe ao Driver a tarefa de coletar os eventos da respectiva tabela, sempre que novos eventos forem detectados.

Tags associados à função **GenSOE** (**elsoe** na configuração por **Strings**) são sempre **reportados a eventos**. Isto significa, conforme já explicado no tópico **Tipos de Dados Definidos pelo Usuário**, que é possível ao Driver retornar vários eventos em uma única operação de leitura, ou seja, em um único intervalo da varredura do Tag.

Isto significa que o Driver retorna o conjunto de eventos (no caso do exemplo anterior, conjuntos de blocos com seis campos de dados e *timestamp*) de uma só vez, o que produz uma sequência de eventos **OnRead** no Tag, um para cada evento (bloco com seis campos de dados e *timestamp*) retornado pelo Driver.

Para instruções detalhadas sobre a maneira correta de tratar Tags reportados a eventos, consulte o tópico **Tags Reportados por Evento** no *Manual do Usuário do E3*. O Manual do Usuário do **Elipse SCADA** também possui um tópico análogo.

Em suma, a forma usual de tratar Tags reportados a eventos é inserir em seu evento **OnRead** o método **WriteRecord** do objeto Histórico previamente associado, garantindo assim a gravação de todos os eventos que cheguem ao Histórico. Neste caso, o Histórico deve ser configurado sem banda morta (propriedade **DeadBand** igual a zero) e desabilitando o histórico por varredura (no **E3**, propriedade **ScanTime** igual a zero). A propriedade **EnableDeadBand** do Tag também deve ser configurada para Falso.

**IMPORTANTE**

Ao ler eventos de memória de massa em Tags reportados a eventos no **E3**, desabilite a banda morta do Tag (propriedade **EnableDeadBand** configurada como Falso) e também no objeto Histórico associado (propriedade **DeadBand** igual a zero), para evitar a perda de eventos com valores próximos. Também é importante desabilitar o histórico por varredura (no **E3**, propriedade **ScanTime** igual a zero). Com isto, garante-se que novos eventos só são armazenados através do método **WriteRecord**, executado no evento **OnRead** do Tag, evitando a duplicação de eventos.

**Otimização e Compatibilidade**

Alguns equipamentos, como os CLPs da marca ATOS, não suportam a leitura em blocos de tipos de dados de estruturas diferentes. Na prática, isto impede que o Driver leia em um bloco único dados de registradores de controle e de eventos. Para coletar eventos de CLP com estas restrições, é preciso desabilitar a opção **Enable Control and Data Registers Grouping** na aba **Gen SOE**.

## Leitura de Registros da Memória de Massa de Medidores ABB MGE 144

Para a leitura de registros de memória de massa de medidores ABB MGE 144, deve-se configurar Tags usando a função especial de leitura **65 03**, conforme descrito neste tópico.

A função especial **65 03** é proprietária da ABB e é praticamente idêntica à função padrão **03** do protocolo (*Read Holding Registers*), diferindo apenas nos dados retornados, referentes à memória de massa do medidor ABB.

Os dados são retornados em um **Word** (como na função **03**), com o *byte order* padrão do protocolo (*big endian*). Com isto, não é necessário habilitar qualquer uma das funções de *swap* (*Swap Bytes*, *Swap Words* ou *Swap DWords*).

O mapa dos registradores do medidor, especificando os dados que podem ser lidos, bem como sua correta configuração, deve ser consultado no manual do medidor fornecido pelo fabricante.

Este Driver também conta com duas funções especiais de escrita específicas deste medidor, as funções **65 01** e **65 02**. Para mais informações sobre estas funções especiais de escrita, consulte o tópico **Funções Especiais** e também o manual do equipamento.

### Configuração por Strings

- **Dispositivo:** "<slave id>:"
- **Item:** "abbmge<endereço>[.<tipo>[<tam. do tipo>]][.<byte order>][<bit>]"

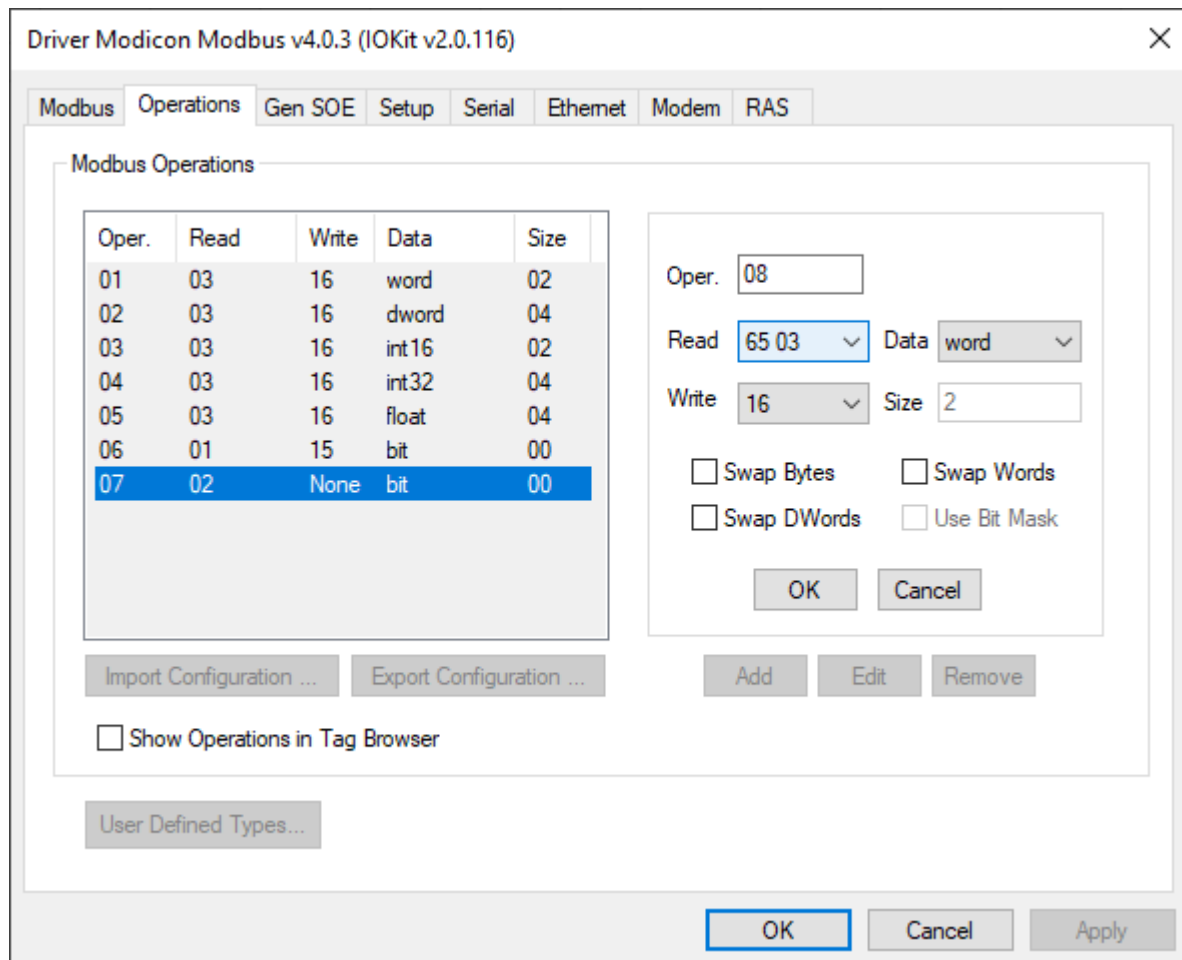
Onde:

- **Endereço:** Endereço do registrador Modbus a ler
- **Tipo:** Tipo de dados. Se omitido, assume o padrão **Word**. Para mais informações, consulte o tópico **Configuração por Strings**
- **Tam. do tipo:** Usado somente em tipos de dados de tamanho variável. Para mais informações, consulte o tópico **Configuração por Strings**
- **Byte order:** Ordenamento dos bytes. Se omitido, assume o padrão do protocolo. Para mais informações, consulte o tópico **Configuração por Strings**
- **Bit:** Máscara de bits. Normalmente omitido (prefira utilizar as máscaras de bit do supervisor). Para mais informações, consulte o tópico **Configuração por Strings**

### Configuração Numérica (Parâmetros N ou B)

- **N1/B1:** Slave Id
- **N2/B2:** Código da **operação** configurada com a função **65 03** (veja a seguir)
- **N3/B3:** Não usado, deixar em 0 (zero)
- **N4/B4:** Endereço do registrador

Para a configuração de Tags numericamente, é preciso antes adicionar uma nova operação com a função **65 03** na aba **Operations** da janela de configurações do Driver, como mostrado na figura a seguir.



Criação de uma operação com a função especial 65 03

## Apêndice

Este apêndice contém os seguintes tópicos:

- Dicas de Otimização
- Dúvidas Mais Frequentes
- Lista de Equipamentos que Comunicam com o Modbus
- Lista de Exceções Padrão do Protocolo
- Limite Máximo para o Tamanho dos Blocos Suportado pelo Protocolo
- Codificação BCD



# Dicas de Otimização

Este tópico enumera algumas dicas de otimização para a comunicação com os dispositivos escravos.

## Dicas de Configuração do Driver para o E3 e o Elipse Power

- Use Superblocos sempre que possível, dando preferência à criação de Tags simples (Tags PLC no **Elipse SCADA**) ao invés de Tags Bloco (veja o tópico **Leitura por Superblocos**).
- Se não for possível usar Superblocos, dê preferência à criação de Tags Bloco, agrupando o maior número possível de variáveis no menor número de blocos (veja o texto sobre o agrupamento manual no tópico **Leitura por Superblocos**).
- Considere as recomendações do artigo *Dicas de Performance para o E3*, no **Elipse Knowledgebase**.
- No caso de redes com alta latência, banda reduzida ou perda de pacotes, leia também o artigo *Configurações de rede do E3 para redes com alta latência, banda reduzida e/ou perda de pacotes*.
- Em redes de alta latência, configure *time-outs* mais elevados, levando em conta a latência esperada. Lembre-se que o tempo de *time-out* só tem efeito no caso de atrasos, não interferindo no desempenho em situações normais de uso.

## Dicas de Configuração do Driver para o Elipse SCADA

- Dê preferência à criação de Tags Bloco, agrupando o maior número possível de variáveis no menor número de blocos (veja o texto sobre o agrupamento manual no tópico **Leitura por Superblocos**).
- Leve em consideração as recomendações do artigo *Desenvolvendo aplicativos para que tenham o melhor desempenho possível*, também disponível no **Elipse Knowledgebase**.
- Em redes de alta latência, configure *time-outs* mais altos, levando em conta a latência esperada. Lembre-se que o tempo de *time-out* só tem efeito no caso de atrasos, não interferindo no desempenho em situações normais de uso.

## Dicas para a Configuração ou Programação do Equipamento

- Se possível, agrupe as variáveis definidas pela aplicação residente (*ladder*) que possuam menor tempo de varredura (*scan*), em endereços contíguos na memória do CLP. O tempo total de varredura dos Tags depende muito da capacidade de agrupamento das variáveis em blocos de comunicação.

# Dúvidas Mais Frequentes

## Particularidades conhecidas do equipamento Twido da empresa Schneider

- A aplicação está tentando ler um valor do tipo **Float**, porém não está conseguindo. O valor mostrado no CLP está totalmente diferente do mostrado na aplicação para o mesmo endereço.
- **Resposta:** Este CLP não utiliza o ordenamento de bytes padrão do protocolo (*big endian*). Deve-se configurar o ordenamento de bytes executando o *swap* (permuta) de **Words**, o que corresponde à opção "b2" na **configuração por Strings**, ou a selecionar a opção **Swap Words** nas configurações das operações da **configuração numérica** (veja o tópico **Aba Operations**).
- A aplicação está tentando ler as entradas e saídas do CLP, porém não está conseguindo.
- **Resposta:** Este equipamento não permite leitura ou escrita nas variáveis de entrada e saída, sendo necessário utilizar variáveis internas ao CLP para realizar esta leitura, isto é, cria-se um espelho das entradas e saídas em uma área onde o Driver consiga acessar. Deve-se ainda ter o cuidado de criar uma rotina no CLP para verificar quando o valor de uma saída for alterada pela aplicação para que ela seja realmente ativada ou desativada no CLP.

## Particularidades conhecidas do equipamento MPC 6006 da empresa Atos - Schneider

- A aplicação está tentando ler um valor do tipo **DWord**, porém o valor correto não está chegando. A aplicação apresenta valores diferentes daqueles que constam no CLP.
- **Resposta:** Consulte o artigo *Utilizando drivers Modbus Master (ASC/RTU/TCP) com controladores ATOS* no **Elipse Knowledgebase**. Se estiver usando a nova configuração por **Strings** (campos **Dispositivo** e **Item**), consulte também o item **Byte Order** do tópico **Configuração por Strings**. Caso esteja utilizando a antiga configuração numérica (parâmetros *N/B*), consulte também o tópico **Aba Operations**, sobretudo o item **Byte Order**.

## Como juntar dois valores do tipo Int16 (no CLP) em um valor do tipo Int32 (na aplicação)?

- Existe um número de 32 bits que está armazenado em dois registradores de 16 bits cada um no CLP. Como fazer para mostrar na tela da aplicação este número como um único registro de 32 bits?
- **Resposta:** Deve-se criar Tags utilizando tipos de dados de 32 bits, como por exemplo os tipos de dados **Float**, **DWord** ou **Int32**. Na configuração do Tag de Comunicação, deve-se informar o primeiro endereço de cada variável no CLP (veja o tópico **Configurando um Tag de Comunicação**). Com isto, o Driver une dois registros de 16 bits do equipamento em um único valor de 32 bits, retornado no campo **Valor** do Tag ou no Elemento do Tag Bloco. Se estiver usando a configuração por **Strings** (campos **Dispositivo** e **Item**), informe o tipo de dados desejado logo após o endereço do registro (veja o tópico **Configuração por Strings**). Caso esteja utilizando a antiga **configuração numérica** (parâmetros *N/B*), é necessário definir operações com tipos de dados de 32 bits. Note que, na janela de configuração (**Aba Operations**), os tipos de dados de 32 bits são sempre mostrados com tamanho (campo **Size**) de quatro bytes (veja o tópico **Tipos de Dados Suportados**).
- A aplicação já está desenvolvida, porém é preciso juntar os valores de dois **Words** em um único Tag.
- **Resposta:** É possível realizar esta união através do uso de scripts, criando um inteiro de 32 bits sem sinal. Para isto, deve-se multiplicar o **Word** que contém a parte mais alta da palavra por 65536 e então somar o **Word** que contém a parte mais baixa da palavra. Por exemplo, **UInt32 = (HighWord × 65536) + LowWord**.
- A aplicação precisa ler valores do tipo **Float**. A função de leitura foi configurada como sendo **03** e escrita **16** com o tipo de dados **Float**. Porém, a aplicação **E3** ou **Elipse Power** mostra um valor que não condiz com o valor que está no equipamento.

- **Resposta:** O protocolo Modbus oficial usa o ordenamento de bytes (*byte order*) no padrão *big endian*, com os bytes mais significativos de cada valor vindo antes. Se o Driver estiver lendo valores absurdos, mesmo com a configuração correta do endereço, é muito provável que o equipamento utilize um *byte order* diferente do padrão do protocolo. Neste caso, é necessário configurar as opções de permuta (*swap*). Se estiver usando a configuração por **Strings** (campos **Dispositivo** e **Item**) consulte o item **Byte Order** do tópico **Configuração por Strings**. Caso esteja utilizando a antiga **configuração numérica** (parâmetros *N/B*), consulte o item **Byte Order** do tópico **Aba Operations** para mais informações sobre como utilizar as opções de *swap*.

## Como fazer para comunicar com mais de um equipamento em uma rede de comunicação Serial?

- Existe mais de um equipamento na rede serial, cada um com endereço único. Como fazer para comunicar com cada um deles?
- **Resposta:** Deve-se ter cuidado apenas com o *Slave Id* de cada **Tag de Comunicação**, pois é neste campo que deve-se indicar com qual equipamento deseja-se comunicar. Em redes seriais RS485, todos os equipamentos escutam ao mesmo tempo as requisições do Driver (há um barramento único), porém apenas o que possuir o *Slave Id* correspondente responde à requisição (não podem haver múltiplos equipamentos com o mesmo *Id*). Na configuração por **Strings**, este valor pode ser fornecido no campo **Dispositivo**, ou no início do campo **Item** (consulte mais informações no tópico **Configuração por Strings**). No caso da **configuração numérica**, este valor é fornecido nos parâmetros *N1/B1* de cada Tag. Pode-se usar as mesmas **operações** para Tags de diversos equipamentos. Uma boa referência para maiores informações sobre a instalação e manutenção de redes seriais é o livro *Serial Port Complete*, de Jan Axelson.
- Existe mais de uma porta serial no computador. Como configurar o Driver para comunicar com os equipamentos que estão ligados em cada uma das portas?
- **Resposta:** Neste caso, como existe mais de um meio físico diferente (Serial 1, Serial 2, etc.), são necessários tantos Drivers de Comunicação quantas portas existirem. As configurações dos **Tags** do Driver podem ser as mesmas para todos os objetos (instâncias) do Driver. A única diferença é que um Driver deve ser configurado para comunicar pela porta Serial 1, outro Driver configurado para comunicar pela porta Serial 2, e assim por diante. As configurações da porta a ser usada são realizadas na aba **Serial** da janela de configurações do Driver (veja o tópico **Propriedades**).

## Como comunicar com mais de um equipamento em uma rede de comunicação Serial com conversor para RS485?

- Uma rede RS485 tem vários equipamentos comunicando através de um conversor RS232-RS485 pela porta Serial. Sempre ocorre a troca de endereço (*Slave ID*), ou seja, quando o Driver vai solicitar dados de outro equipamento, ocorre um *time-out*. Após retentar a mesma mensagem, o equipamento responde normalmente. Existe alguma forma de evitar este *time-out* durante a troca do endereço (*Slave ID*)?
- **Resposta:** Alguns conversores RS232-RS485 requerem um intervalo de tempo para chavearem, ou seja, comutarem do modo de transmissão para recepção, ou vice-versa. Para contornar esta limitação, pode-se utilizar a opção **Inter-frame delay** na aba **Serial** do IOKit, disponível na **janela de configurações**. Este campo define um intervalo de tempo entre mensagens. O valor exato do intervalo depende do conversor utilizado mas, em caso de incerteza, recomenda-se iniciar experimentando valores entre 50 ms e 300 ms.

### NOTA

A opção **Inter-frame delay** do IOKit pode trazer significativo prejuízo de performance em algumas aplicações, devendo ser utilizada apenas quando absolutamente necessário. Certifique-se de que o conversor está em boas condições, e se ele de fato exige a utilização deste *delay*. Se necessário, contate o suporte do fabricante.

## Como fazer para comunicar com mais de um equipamento em uma rede de comunicação Ethernet?

- Existe mais de um equipamento ligado em uma rede Ethernet, cada um com endereço IP único. Como fazer para comunicar com cada um deles?
- **Resposta:** Atualmente, para cada endereço IP, é necessário tantos Drivers de Comunicação quantos endereços IP deseja-se comunicar. A configuração referente aos **Tags** do Driver pode ser a mesma para todos os Drivers. A única diferença é que um Driver deve ser configurado para comunicar com o endereço IP 1 (um), outro Driver configurado para comunicar com o endereço IP 2 (dois), e assim por diante. O parâmetro *Slave Id* pode ainda ser utilizado em modo **Modbus TCP** para diferenciar equipamentos conectados a um *gateway* Modbus Ethernet / RS485 no mesmo endereço IP. Note que este *gateway* não apenas deve permitir a interconexão entre redes Ethernet e seriais, mas também converter *frames* ModbusTCP para os modos seriais suportados pelos equipamentos (**ModbusRTU** ou **ModbusASC**). O endereço IP deve ser configurado na aba **Ethernet** do IOKit, na **janela de configurações** do Driver.

#### DICA

Evite usar o modo **RTU** ou **ASC** do protocolo encapsulado em meio **TCP/IP**. Caso seja necessário encapsular a comunicação serial de equipamentos que utilizem o **Modbus RTU** em **TCP/IP**, existem *gateways* disponíveis no mercado que não somente encapsulam a comunicação serial em **Ethernet TCP/IP** (camadas física, de rede e de transporte), como também convertem o **Modbus RTU** em **Modbus TCP** (camada de aplicação). Em último caso, se for inevitável a utilização de **Modbus RTU** em meio **Ethernet TCP/IP**, não deixe de habilitar a opção **Reconnect after Timeout**, descrita no tópico **Aba Modbus**.

### Software Simulador Modbus

- Existe algum software que simule o protocolo Modbus e que possa ser utilizado para testes junto com o Driver?
- **Resposta:** Sim, existem diversas alternativas. A Elipse Software disponibiliza em seu site uma versão gratuita (demo) do *Elipse Modbus Simulator*, que permite simular os recursos mais básicos do protocolo. Há também a possibilidade de usar o Driver Modbus Slave da **Elipse Software** como emulador. Outra possibilidade é o software Modsim, uma das mais antigas e conhecidas alternativas para emular um dispositivo Modbus escravo. Este simulador pode ser adquirido em <http://www.win-tech.com/html/modsim32.htm>. Além deste, existe também a alternativa gratuita do **Free Modbus PLC Simulator**, disponível em [<%LINK\\_PLC\\_SIMULATOR%>](#). Existem ainda outras opções e uma lista de outros softwares pode ser encontrada no site da *Modbus.org*.

### Como configurar o parâmetro N4/B4 dos Tags de Comunicação?

- Qual endereço utilizar no parâmetro *N4/B4* do Tag de Comunicação?
- **Resposta:** Este endereço varia de equipamento para equipamento. Para saber qual o endereço exato a ser utilizado, consulte o manual do equipamento ou entre em contato com o suporte técnico. O tópico **Dicas de Endereçamento** deste manual contém algumas dicas de convenções comuns usadas por muitos fabricantes.

### Quando utilizar os controles de RTS e DTR (que aparecem na aba Serial da janela de configurações do Driver)?

- O equipamento está comunicando diretamente na porta serial RS232 do computador. Como devo configurar os controles **RTS** e **DTR**?
- **Resposta:** Deve-se consultar a documentação do equipamento ou o suporte do fabricante para saber a configuração correta.
- O equipamento está comunicando através de um conversor RS232-RS485 ligado à porta serial RS232 do computador. Como configurar os controles **RTS** e **DTR**?

- **Resposta:** Na comunicação com equipamentos usando conversores RS232-RS485, tais configurações dependem do conversor. O equipamento (*Slave*) não tem influência, já que estes sinais só existem no lado serial RS232, não tendo equivalentes no meio serial RS485. O controle **RTS** é geralmente utilizado em conversores mais antigos para o chaveamento entre os modos de transmissão e recepção (o RS485 é *half-duplex*), devendo nestes casos em geral ser configurado no modo **Toggle** (existem alguns raros equipamentos que exigem outras configurações). Na maioria dos conversores mais recentes, entretanto, o chaveamento entre transmissão e recepção é automático, e estes sinais em geral não são mais usados, podendo ser ignorados. Em caso de dúvidas, consulte o manual do conversor ou o suporte do fabricante.

## Quando utilizar as funções Swap Bytes, Swap Words e Swap DWords?

Estas opções devem ser usadas para tipos de dados de 16, 32 ou 64 bits, cuja ordem dos bytes do valor fornecido pelo equipamento não corresponde à ordem padrão do protocolo Modbus, onde os bytes mais significativos vêm sempre antes (padrão *big endian*, também chamado *Motorola*). Se o Driver está lendo valores absurdos ou diferentes dos valores armazenados no CLP, é possível que esteja utilizando um *byte order* diferente do padrão do protocolo. Para mais informações, consulte o item **Byte Order** do tópico **Configuração por Strings**, ou caso esteja usando a antiga configuração numérica (parâmetros *N/B*), consulte o item **Byte Order** do tópico **Aba Operations**. Também recomenda-se consultar a documentação do equipamento.

- A aplicação está tentando ler um valor **Word**, porém o valor vem diferente do que está configurado no CLP. Se no CLP é configurado o valor "1" (um), na aplicação aparece o valor "256".
- **Resposta:** O valor 1 (um) em hexadecimal é 0001H e o valor 256 em hexadecimal corresponde a 0100H. O equipamento possui um *byte order* diferente do padrão do protocolo. Deve-se habilitar a opção **Swap Bytes** (opção "b1" na **configuração por Strings**) para ler o valor correto.
- A aplicação tem um Tag configurado para ler um valor **DWord**, mas o valor lido pela aplicação é diferente do valor armazenado no CLP. Ao atribuir o valor "258", por exemplo, ao registro no CLP, na aplicação aparece o valor absurdo "16908288".
- **Resposta:** O valor 258 em hexadecimal é 00000102H e o valor 16908288 em hexadecimal corresponde a 01020000H. O equipamento possui um *byte order* diferente do padrão do protocolo, onde o **Word** menos significativo vem antes. Neste caso, deve-se habilitar a opção **Swap Words** (opção "b2" na **configuração por Strings**) para ler o valor correto.

## Como ler corretamente tipos de dados Float em CLPs WEG TPW-03?

- **Resposta:** Na configuração dos **Tags de Comunicação**, deve-se habilitar a opção de ordenamento de bytes **Swap Words**, correspondente à opção "b2" na **configuração por Strings**. Caso esteja usando a antiga configuração numérica (parâmetros *N/B*), consulte o item **Byte Order** do tópico **Aba Operations**.

## Particularidades conhecidas dos equipamentos da família ABB Advant Controller 31 Series 90 (por exemplo, ABB 07KT97)

- A aplicação no E3 ou no Elipse Power está tentando ler registros ou bits do CLP, mas sempre ocorrem erros.
- **Resposta:** Os equipamentos desta série não permitem o uso de Superblocos do E3 ou do Elipse Power por dois motivos:
  - Existem discontinuidades no mapa de endereços de registradores dos equipamentos, com intervalos de endereços não definidos.

- O tamanho máximo do **PDU** é diferente do estabelecido pelo padrão do protocolo, sendo definido como um tamanho que suporte 96 **Words** ou **Bits**. Uma vez que o protocolo agrupa oito bits em cada byte de dados, isto resulta em tamanhos máximos de **PDU** diferentes para as funções de leitura de **Bits** e **Words**, o que impossibilita o uso da customização do tamanho máximo do **PDU** permitida pelo Driver, que não permite configurar limites diferentes para cada função do protocolo.
- **Solução:** Siga estes passos:
  1. Desabilite a leitura por Superblocos, configurando a propriedade **EnableReadGrouping** do Driver para Falso.
  2. Dê preferência à definição de Tags Bloco, agrupando o maior número possível de variáveis no menor número de Blocos, respeitando o limite do equipamento de 96 **Words** ou 96 **Bits** em cada Bloco (para mais informações, leia a seção sobre **Agrupamento Manual** no tópico **Leitura por Superblocos**).

#### NOTA

Pode ainda ser possível utilizar o agrupamento automático (Superblocos) se não for preciso ler **Words** e **Bits** no mesmo objeto Driver, dependendo obviamente do intervalo de endereços que se precise ler (mais especificamente, se este intervalo possui ou não descontinuidades). Neste caso, de qualquer forma, é necessário configurar a propriedade **Customize Max. PDU Size** na **Aba Modbus**, de acordo com o limite de 96 **Words** ( $96 \times 2 = 192$  bytes) ou 96 Bits ( $96 \div 8 = 12$  bytes). Tal possibilidade pode ser avaliada com cuidado, caso a caso, pelo desenvolvedor da aplicação.

### A aplicação está tentando ler valores do tipo de dados Float e a seguinte mensagem aparece no log do Driver: "Warning: denormalized float number! Returning zero". O que fazer?

- **Resposta:** O aparecimento desta mensagem não indica erro de comunicação ou de configuração. Recomenda-se apenas que o usuário verifique na programação do CLP por que ele está retornando valores não normalizados.
- **Informações Adicionais:** Tal mensagem indica que o equipamento enviou ao Driver um valor de ponto flutuante (**Float**) no formato **IEEE 754**, porém não normalizado. Tais valores podem resultar de operações aritméticas com resultados que extrapolem as possibilidades de representação deste formato, como as condições de *overflow*, *underflow*,  $+\infty$  e  $-\infty$ , etc. Os valores não normalizados estão previstos na norma IEEE 754, não devendo em tese gerar problemas para o Driver ou para a aplicação. Entretanto, devido à detecção de erros no passado relacionada a hardwares específicos, o Driver passou a retornar 0 (zero) para a aplicação ao receber valores não normalizados do equipamento, registrando esta mensagem em log.

## Lista de Equipamentos que Comunicam com o Modbus

A tabela a seguir contém uma lista de equipamentos, separados por fabricante, para os quais já existe alguma experiência na comunicação com o protocolo Modbus.

Para uma lista mais completa de equipamentos já validados com o protocolo, consulte o *Modbus Device Directory*, mantido pela Organização Modbus.

#### Equipamentos que comunicam com o Modbus

FABRICANTE	EQUIPAMENTO
ABB	<ul style="list-style-type: none"> <li>• ETE30</li> <li>• MGE 144</li> <li>• KT97</li> <li>• KT98</li> </ul>
Altus	<ul style="list-style-type: none"> <li>• Praticamente todos os equipamentos da Altus possuem Modbus, exceto alguns modelos da linha Piccolo</li> </ul>

FABRICANTE	EQUIPAMENTO
<b>Areva</b>	<ul style="list-style-type: none"> <li>• MiCOM P127</li> <li>• Relé P632</li> </ul>
<b>Atos</b>	Os CLPs ATOS suportam o protocolo <b>Modbus RTU</b> com pequenas variações relativas ao tamanho máximo dos <i>frames</i> e <i>byte order</i> . Para mais informações sobre as variações, consulte o artigo <i>Utilizando o driver Modbus Master com controladores ATOS</i> , no <b>Elipse Knowledgebase</b>
<b>BCM</b>	<ul style="list-style-type: none"> <li>• BCM1088</li> <li>• BCM1086</li> <li>• BCM-GP3000</li> <li>• BCM2085</li> </ul>
<b>Ciber Brasil</b>	<ul style="list-style-type: none"> <li>• Medidor Multivariável de Grandezas elétricas UDP200</li> <li>• Medidor Multivariável de Grandezas elétricas UDP600</li> </ul>
<b>Contemp</b>	<ul style="list-style-type: none"> <li>• CPM45</li> </ul>
<b>Deep Sea</b>	<ul style="list-style-type: none"> <li>• DSE5210</li> <li>• DSE5310</li> <li>• DSE5310M</li> <li>• DSE5320</li> <li>• DSE5510</li> <li>• DSE5510M</li> <li>• DSE5520</li> <li>• DSE7310</li> <li>• DSE7320</li> </ul>
<b>Embrasul</b>	<ul style="list-style-type: none"> <li>• Medidor MD4040</li> </ul>
<b>Eurotherm</b>	<ul style="list-style-type: none"> <li>• 2500 Intelligent Data Acquisition and Precision Multi-Loop PID Control</li> </ul>
<b>Fatek</b>	<ul style="list-style-type: none"> <li>• FB - 14MCU</li> </ul>
<b>GE</b>	<ul style="list-style-type: none"> <li>• GE PAC RX7</li> <li>• GE GEDE UPS</li> </ul>
<b>Gefran</b>	<ul style="list-style-type: none"> <li>• CLP Gefran Gilogk II</li> <li>• Gefran 600RDR21</li> </ul>
<b>Hitachi</b>	<ul style="list-style-type: none"> <li>• Hitachi HDL17264</li> </ul>
<b>Honeywell</b>	<ul style="list-style-type: none"> <li>• PLC HC-900</li> </ul>
<b>Horner APG</b>	<ul style="list-style-type: none"> <li>• Equipamentos da série XLe/XLt all-in-one control devices</li> </ul>
<b>Koyo</b>	<ul style="list-style-type: none"> <li>• CLP CPU 260</li> </ul>
<b>Kron</b>	<ul style="list-style-type: none"> <li>• Medidor MKM-120</li> </ul>
<b>LG</b>	<ul style="list-style-type: none"> <li>• DMT40U</li> </ul>
<b>LG / LSIS</b>	<ul style="list-style-type: none"> <li>• LG Master K120 S</li> <li>• LG XGB - XBM</li> </ul>

FABRICANTE	EQUIPAMENTO
Moeller	<ul style="list-style-type: none"> <li>XC100 - Porta Serial 232</li> <li>XC200 - Porta Serial 232/422/485 (módulo de comunicação XIO-SER) e Porta Ethernet</li> <li>XV200 - Porta Serial 232 e Porta Ethernet</li> <li>XVH300 - Porta Serial 232 e Porta Ethernet</li> <li>XV400 - Porta Serial 232 e Porta Ethernet</li> </ul>
Novus	<ul style="list-style-type: none"> <li>N1100</li> <li>N1500</li> <li>N2000</li> <li>N3000</li> </ul>
Schneider	<ul style="list-style-type: none"> <li>Twido</li> <li>A340</li> <li>CLP M340</li> <li>Série Premium</li> <li>Conversores de frequência e <i>soft starter</i></li> <li>Disjuntores de MT e BT</li> <li>Relés de proteção de BT e MT</li> <li>Relés SEPAM Séries 20, 40 e 80</li> </ul>
Telemecanique	<ul style="list-style-type: none"> <li>Controladores Zelio Logic com final "BD"</li> </ul>
Unitronics	<ul style="list-style-type: none"> <li>V120</li> </ul>
Weg	<ul style="list-style-type: none"> <li>TP 03</li> <li>Clic 02</li> </ul>
Yaskawa	<ul style="list-style-type: none"> <li>V1000</li> </ul>

## Lista de Exceções Padrão do Protocolo

A tabela a seguir lista as exceções padrão, definidas pela especificação do protocolo Modbus (versão 1.1b).

As exceções são registradas no log do Driver, sempre que detectadas, e podem ser lidas pela aplicação através da **Leitura do Código da Última Exceção**.

Note que, além das exceções listadas aqui, o equipamento pode definir outras exceções proprietárias. Neste caso, espera-se que estas exceções sejam descritas na documentação do fabricante do equipamento.

### Códigos de exceção padronizados pelo protocolo Modbus

CÓDIGO (EM HEXADECIMAL)	NOME	SIGNIFICADO
01	ILLEGAL FUNCTION	O código de função recebido não é válido. Isto pode indicar que a função não está implementada ou que o Escravo encontra-se em um estado inadequado para processá-la



CÓDIGO (EM HEXADECIMAL)	NOME	SIGNIFICADO
02	ILLEGAL DATA ADDRESS	O endereço de dados recebido não é um endereço válido. Mais especificamente, a combinação do endereço de referência e a quantidade de dados a serem transferidos é inválida
03	ILLEGAL DATA VALUE	O valor presente na requisição do Mestre não é válido. Isto indica uma falha na estrutura de dados remanescente de uma requisição complexa, como quando o tamanho informado para o bloco de dados não está correto. Esta exceção não indica que os valores submetidos para escrita estejam fora do escopo esperado pela aplicação, uma vez que tal informação não é acessível ao protocolo
04	SLAVE DEVICE FAILURE	Ocorreu um erro irreversível durante o processamento da função solicitada
05	ACKNOWLEDGE	Usado com comandos de programação. O Escravo aceitou a mensagem e a está processando, mas este processamento demanda um longo tempo. Esta exceção previne um <i>time-out</i> no Mestre. O fim da requisição deve ser testado por um processo de <i>polling</i>
06	SLAVE DEVICE BUSY	Usado com comandos de programação. Indica que o Escravo está processando um outro comando de longa duração e que a solicitação deve ser retransmitida mais tarde, quando o Escravo estiver novamente disponível
08	MEMORY PARITY ERROR	Usado em conjunto com as funções <b>20</b> e <b>21</b> , <i>reference type 6</i> , para indicar que a área estendida de arquivos falhou em um teste de consistência. O equipamento Escravo pode estar precisando de manutenção
0A	GATEWAY PATH UNAVAILABLE	Usado em conjunto com <i>gateways</i> , para indicar que o <i>gateway</i> não foi capaz de alocar um caminho interno para o processamento da solicitação. Geralmente indica que o <i>gateway</i> está desconfigurado ou sobrecarregado

CÓDIGO (EM HEXADECIMAL)	NOME	SIGNIFICADO
0B	GATEWAY TARGET DEVICE FAILED TO RESPOND	Usado em conjunto com <i>gateways</i> , para indicar que não foi recebida nenhuma resposta do equipamento de destino. Geralmente indica que o equipamento não está presente na rede

## Limite Máximo para o Tamanho dos Blocos Suportado pelo Protocolo

Neste tópico são apresentados os limites máximos de tamanho de bloco suportados pelo protocolo Modbus, na atual versão 1.1b de sua especificação (veja a especificação no *site oficial do protocolo*).

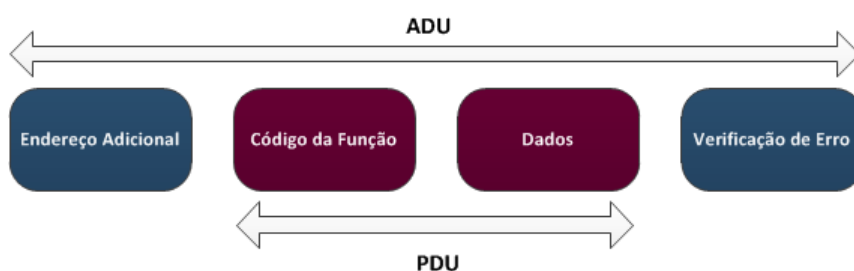
Note que, devido aos recursos de **Leitura por Superblocos** e **Partição Automática de Blocos**, presentes na versão atual do Driver, dificilmente o usuário necessita levar em conta estes limites em uma aplicação, uma vez que o Driver já realiza automaticamente as otimizações necessárias no momento da comunicação.

Entretanto, uma vez que existem equipamentos que não suportam os limites padrão estabelecidos pelo protocolo, pode ser necessário ao usuário conhecer os limites do protocolo, e sobretudo saber avaliar os limites do equipamento, caso seja obrigado a realizar o agrupamento de forma manual (veja o tópico **Leitura por Superblocos**). Nestes casos, a informação deste tópico pode se mostrar útil.

### Limites Suportados pelo Protocolo

O protocolo Modbus define uma unidade de dados simples denominada **PDU** (*Protocol Data Unit*), que se mantém inalterada nos diversos modos do protocolo e nas diversas camadas de comunicação.

O *frame* de comunicação completo, incluindo a PDU e os demais campos adicionais de cabeçalho, é chamado **ADU** (*Application Data Unit*).



Limites suportados pelo protocolo

Segundo a especificação do protocolo, o *frame* Modbus completo (ADU) pode ter uma PDU com tamanho máximo de 253 bytes.

Sendo assim, dependendo do tipo de dado ou função Modbus que é utilizado na comunicação, o protocolo impõe os limites de elementos do bloco em cada comunicação descritos na tabela a seguir.

### Limites de Elementos de Bloco

FUNÇÃO MODBUS	DESCRIÇÃO	LIMITE
03, 04	Leitura de múltiplos registros de 16 bits	125 registros (250 bytes)
16	Escrita de múltiplos registros de 16 bits ( <i>Holding Registers</i> )	123 registros (247 bytes)

FUNÇÃO MODBUS	DESCRIÇÃO	LIMITE
<b>01, 02</b>	Leitura de múltiplos bits	2000 bits (250 bytes)
<b>15</b>	Escrita de múltiplos bits	1968 bits (247 bytes)
<b>20</b>	Leitura de registros de arquivo	124 registros (248 bytes)
<b>21</b>	Escrita de registros de arquivo	122 registros (244 bytes)

Mais informações podem ser obtidas no *site oficial do protocolo*.

O artigo *KB-23112: Tamanho ideal de um Bloco de Comunicação usando o Driver Modbus* no **Elipse Knowledgebase** apresenta uma síntese das questões relativas ao agrupamento de Tags e dimensionamento de blocos neste Driver, discutidas neste e em outros tópicos.

## Codificação BCD

A **Codificação BCD** (*Binary-Coded Decimal* ou *Decimal Codificado em Binário*) foi originalmente concebida para contornar limitações quanto ao número máximo de dígitos passíveis de serem representados nos formatos mais tradicionais de armazenamento de valores. Formatos como a representação de números reais em ponto flutuante mostram-se normalmente aceitáveis para cálculos matemáticos e científicos. Porém, erros de aproximação causados pela existência de algarismos que não possam ser representados por problemas de *overflow* ou *underflow* podem não ser admissíveis em certas aplicações, como em procedimentos financeiros. Para superar este tipo de limitação, foi desenvolvida a codificação BCD, que permite a representação de números até o último algarismo.

Nessa representação, cada algarismo decimal é representado isoladamente em formato binário, sem limitações no que se refere ao número de algarismos.

A tabela a seguir mostra os algarismos decimais e seus valores correspondentes em BCD (valores em binário).

**Algarismos decimais em codificação BCD**

DECIMAL	BCD	DECIMAL	BCD
0	0000b	5	0101b
1	0001b	6	0110b
2	0010b	7	0111b
3	0011b	8	1000b
4	0100b	9	1001b

A fim de melhorar a eficiência desta codificação, é comum representar-se dois algarismos por byte. Note que, na tabela acima, cada dígito decimal requer apenas quatro bits, ou meio byte, para a sua representação.

Tal representação com dois dígitos em cada byte é chamada de **BCD Comprimido** (*Packed BCD*), e é a representação utilizada por este Driver, ou seja, os pacotes enviados por este Driver com valores BCD utilizam um byte de dados para cada dois algarismos do valor decimal representado. Por isto o campo **Size**, no caso de tipos de dados **BCD**, deve ser definido como a metade do número máximo de algarismos representados nos valores a serem lidos ou escritos.

### Exemplo

Como exemplo, suponha que se pretenda enviar o valor 84 em decimal (0x54 em formato hexadecimal), usando a codificação BCD comprimido em um byte, o formato usado por este Driver.

O primeiro passo é separar os dois dígitos decimais que compõem o valor em sua representação decimal:

- **Dígito 1:** 8
- **Dígito 2:** 4

Se fossemos enviar o valor ao equipamento sem a codificação BCD, o valor enviado ao protocolo seria o próprio valor 84, que seria representado em formato hexadecimal pelo valor 0x54, ou ainda 01010100b em formato binário.

Usando o formato BCD comprimido, entretanto, representaremos os dois dígitos decimais separadamente em cada metade, ou *nibble*, do byte a ser enviado:

- **BCD:** 0x84 ou 1000100b

Note que, se interpretássemos por engano este valor 0x84 em formato BCD como um valor em formato hexadecimal sem esta codificação, e este valor fosse convertido para decimal, obteríamos o valor 132, sem significado algum.

A tabela a seguir apresenta mais alguns exemplos de valores decimais entre 0 (zero) e 99 e suas respectivas representações no formato BCD Comprimido em um byte, apresentados nos formatos hexadecimal e binário.

#### Algarismos decimais em codificação BCD Comprimido

DECIMAL	HEXADECIMAL	BCD (HEXADECIMAL)	BCD (BINÁRIO)
10	0x0A	0x10	00010000b
0	0x00	0x00	00000000b
99	0x63	0x99	10011001b
81	0x51	0x81	10000001b
45	0x2D	0x45	01000101b

#### Exemplo de Parametrização para o formato BCD:

1) Leitura do valor 11223344 em formato BCD

Note que o inteiro 11223344 possui "8 algarismos", sendo assim, no dialog de propriedades do driver Modbus, devemos criar uma nova "Operação", definindo o campo "Data" como tipo BCD e o campo "Size" com tamanho 4, ou seja, os 8 algarismos do valor BCD devem ser dimensionados para bytes, agrupando os algarismos sempre aos pares (11 22 33 44) para saber quantos bytes serão necessários para representá-lo.

#### Leitura de um valor com codificação BCD Comprimido

MEMÓRIA DO EQUIPAMENTO	VALOR DA TAG	NOTAS
Registro 1 = 0x1122 Registro 2 = 0x3344	11223344	O valor 11223344 necessita de 4 bytes para ser representado em formato BCD, sendo necessário ocupar 2 registros Modbus na memória do equipamento.

## Documentação das Interfaces de Comunicação

Esta seção contém a documentação das Interfaces de Comunicação referentes ao Driver **Modbus**.

## Configurações do Driver

A configuração das Interfaces de Comunicação é realizada na caixa de diálogo de configuração do Driver. Para acessar a configuração da caixa de diálogo no **E3** (versão 1.0), siga estes passos:

1. Clique com o botão direito do mouse no objeto Driver (IODriver).
2. Selecione o item **Propriedades** no menu contextual.
3. Selecione a aba **Driver**.
4. Clique em **Outros parâmetros**.

No **E3** versão 2.0 ou posterior, clique em **Configurar o driver**  na barra de ferramentas do Driver. No **Eclipse SCADA**, siga estes passos:

1. Abra o Organizer.
2. Selecione o Driver na árvore do Organizer.
3. Clique em **Extras** na aba **Driver**.

Atualmente, as Interfaces de Comunicação permitem que apenas uma conexão seja aberta para cada Driver. Isto significa que, se for necessário o acesso a duas portas seriais, é preciso adicionar dois Drivers na aplicação e configurar cada um destes Drivers para cada porta serial.

## Caixa de Diálogo de Configuração

A caixa de diálogo das Interfaces de Configuração permitem configurar a conexão de I/O que é utilizada pelo Driver. Esta caixa de diálogo contém as abas **Setup**, **Serial**, **Ethernet**, **Modem** e **RAS**, descritas nos tópicos a seguir. Se um Driver não implementa uma conexão de I/O específica, a respectiva aba não está disponível para configuração. Alguns Drivers podem conter abas adicionais (específicas para cada Driver) na caixa de diálogo de configuração.

## Aba Setup

A aba **Setup** contém a configuração geral do Driver. A aba é dividida em três partes distintas:

- **Configurações gerais:** Configurações da camada física do Driver, *time-out* e modo de inicialização
- **Connection management:** Configurações de como a Interface de Comunicação mantém a conexão e qual a política de recuperação em caso de falha
- **Logging options:** Controla a geração dos arquivos de log

The screenshot shows the 'Setup' tab of a configuration window. It contains three main sections: 'Physical Layer' with a dropdown set to 'Serial' and a 'Start driver OFFLINE' checkbox; 'Timeout' with a text box set to '1000' and 'ms'; and 'Connection management' with a 'Mode' dropdown set to 'Automatic (managed by the driver)', a 'Retry failed connection every' checkbox checked with a value of '20' seconds, a 'Give up after' checkbox unchecked with a value of '1' failed retries, and a 'Disconnect if non-responsive for' checkbox unchecked with a value of '0' seconds. Below these is a 'Logging Options' section with a 'Log to File' checkbox unchecked and an empty text box.

**Aba Setup**

### Opções gerais da aba Setup

OPÇÃO	DESCRIÇÃO
<b>Physical Layer</b>	Selecione a interface física em uma lista. As opções são <b>Serial</b> , <b>Ethernet</b> , <b>Modem</b> e <b>RAS</b> . A interface selecionada deve ser configurada na sua aba específica.
<b>Timeout</b>	Configure o <i>time-out</i> , em milissegundos, para a camada física. Esta é a medida de tempo que a interface de I/O aguarda para a recepção de um byte (qualquer byte do <i>buffer</i> de recepção).
<b>Start driver OFFLINE</b>	Selecione esta opção para que o Driver inicie em modo <b>Offline</b> (parado). Isto significa que a interface de I/O não é criada até que se configure o Driver em modo <b>Online</b> (utilizando-se um Tag na aplicação). Este modo possibilita a configuração dinâmica da interface de I/O em tempo de execução. Veja o tópico <b>Trabalhando em Modo Offline</b> para maiores detalhes.

### Opções para o grupo Connection management

OPÇÃO	DESCRIÇÃO
<b>Mode</b>	Seleciona o modo de gerenciamento de conexão. Selecionar a opção <b>Automatic</b> permite que o Driver gerencie a conexão automaticamente, como especificado nas opções seguintes. Selecionar a opção <b>Manual</b> permite que a aplicação gerencie a conexão completamente. Veja o tópico <b>Estados do Driver</b> para maiores detalhes.
<b>Retry failed connection every ... seconds</b>	Selecione esta opção para habilitar a retentativa de conexão do Driver em um determinado intervalo, em segundos. Se a opção <b>Give up after failed retries</b> não estiver selecionada, o Driver continua retentando até que a conexão seja efetuada, ou que a aplicação seja parada.
<b>Give up after ... failed retries</b>	Habilite esta opção para definir um número máximo de retentativas de conexão. Quando o número especificado de tentativas consecutivas de reconexão for atingido, o Driver vai para o modo <b>Offline</b> , assumindo que um problema de hardware foi detectado. Se o Driver estabelece uma conexão com sucesso, o número de retentativas sem sucesso é zerado. Se esta nova conexão é perdida, então o contador de retentativas inicia do zero.
<b>Disconnect if non-responsive for ... seconds</b>	Habilite esta opção para forçar o Driver a se desconectar se nenhum byte chegou à interface de I/O no <i>time-out</i> especificado, em segundos. Este <i>time-out</i> deve ser maior que o <i>time-out</i> configurado na opção <b>Timeout</b> .

#### Opções para o grupo Logging Options

OPÇÃO	DESCRIÇÃO
<b>Log to File</b>	<p>Habilite esta opção e configure o nome do arquivo onde o log é escrito. Arquivos de log podem ser bem extensos, portanto utilize esta opção por curtos períodos de tempo, apenas para o propósito de testes e depurações.</p> <p>Caso se utilize a macro <b>%PROCESS%</b> no nome do arquivo de log, esta é substituída pelo ID do processo atual. Esta opção é particularmente útil ao se utilizar várias instâncias do mesmo Driver no <b>E3</b>, permitindo assim que cada instância gere um arquivo separado de log. Por exemplo, ao configurar esta opção como <b>c:\e3logs\drivers\sim_%PROCESS%.log</b>, gera-se o arquivo <b>c:\e3logs\drivers\sim_00000FDA.log</b> para o processo <b>0FDAh</b>.</p> <p>Pode-se também utilizar a macro <b>%DATE%</b> no nome do arquivo. Neste caso é gerado um arquivo de log por dia (no formato <b>aaaa_mm_dd</b>). Por exemplo, ao configurar esta opção como <b>c:\e3logs\drivers\sim_%DATE%.log</b>, gera-se o arquivo <b>c:\e3logs\drivers\sim_2005_12_31.log</b> no dia 31 de dezembro de 2005 e o arquivo <b>c:\e3logs\drivers\sim_2006_01_01.log</b> no dia primeiro de janeiro de 2006.</p>



Aba Serial

Utilize esta aba para configurar os parâmetros da Interface **Serial**.

Serial

Port:
COM1
Baud rate:
9600
Data bits:
8 data bits
Parity:
None
Stop bits:
1 stop bit
☐ Enable 'ECHO' suppression

Handshaking

DTR control:
OFF
RTS control:
OFF
☐ Wait for CTS before send
CTS timeout:
0 ms
Delay before send:
0 ms
Delay after send:
0 ms

Inter-byte delay (microseconds):
0 µs
Inter-frame delay (milliseconds):
0 ms

Aba Serial

Opções gerais da aba Serial

OPÇÃO	DESCRIÇÃO
Port	Selecione a porta serial a partir da lista (de <b>COM1</b> até <b>COM4</b> ) ou digite o nome da porta serial no formato <b>COMn</b> (por exemplo, "COM15"). Ao digitar o nome da porta manualmente, a caixa de diálogo aceita apenas nomes de portas começando com a expressão "COM".
Baud rate	Selecione o <i>baud rate</i> a partir da lista ( <b>1200</b> , <b>2400</b> , <b>4800</b> , <b>9600</b> , <b>19200</b> , <b>38400</b> , <b>57600</b> ou <b>115200</b> ) ou digite o <i>baud rate</i> desejado (por exemplo, 600).
Data bits	Selecione 7 ou 8 bits de dados a partir da lista.
Parity	Selecione a paridade a partir da lista ( <b>None</b> , <b>Even</b> , <b>Odd</b> , <b>Mark</b> ou <b>List</b> ).
Stop bits	Selecione o número de stop bits a partir da lista ( <b>1</b> , <b>1.5</b> ou <b>2</b> stop bits).
Enable 'ECHO' suppression	Habilite esta opção para remover o eco recebido após a Interface de Comunicação enviar dados pela porta serial. Se o eco não for igual aos bytes recém enviados, a Interface de Comunicação aborta a comunicação.

OPÇÃO	DESCRIÇÃO
<b>Inter-byte delay (microseconds)</b>	Define uma espera entre cada byte transmitido pela Interface de Comunicação, em milionésimos de segundo (1000000 igual a um segundo). Esta opção deve ser utilizada com esperas pequenas (menos de um milissegundo).
<b>Inter-frame delay (milliseconds)</b>	Define uma espera entre pacotes enviados ou recebidos pela Interface de Comunicação, em milésimos de segundo (1000 igual a um segundo). Esta espera é aplicada caso a Interface de Comunicação envie dois pacotes consecutivos, ou entre um pacote recebido e o próximo envio.

O grupo **Handshaking** configura o uso dos sinais **RTS**, **CTS** e **DTR** no processo de *handshaking* (controlar quando o dado pode ser enviado ou recebido através da linha serial). Na maioria das vezes, configurar a opção **DTR control** para **ON** e a opção **RTS control** para **Toggle** funciona tanto com linhas seriais RS232 quanto com linhas seriais RS485.

#### Opções disponíveis no grupo Handshaking

OPÇÃO	DESCRIÇÃO
<b>DTR control</b>	Selecione <b>ON</b> para deixar o sinal <b>DTR</b> sempre ligado enquanto a porta serial está aberta. Selecione <b>OFF</b> para desligar o sinal <b>DTR</b> enquanto a porta serial está aberta. Alguns equipamentos exigem que o sinal <b>DTR</b> esteja ligado para permitir a comunicação.
<b>RTS control</b>	Selecione <b>ON</b> para deixar o sinal <b>RTS</b> sempre ligado enquanto a porta serial está aberta. Selecione <b>OFF</b> para desligar o sinal <b>RTS</b> enquanto a porta serial está aberta. Selecione <b>Toggle</b> para ligar o sinal <b>RTS</b> enquanto se envia os bytes através da porta serial, e desligá-lo quando não se está enviando bytes e, portanto, habilitando a recepção.
<b>Wait for CTS before send</b>	Disponível apenas quando a opção <b>RTS control</b> está configurada para <b>Toggle</b> . Utilize esta opção para forçar o Driver a verificar o sinal <b>CTS</b> antes de enviar os bytes através da porta serial, após ligar o sinal de <b>RTS</b> . Neste modo o sinal <b>CTS</b> é tratado como um <i>flag</i> de permissão para envio.
<b>CTS timeout</b>	Determina o tempo máximo, em milissegundos, que o Driver aguarda pelo sinal de <b>CTS</b> depois de ligar o sinal de <b>RTS</b> . Se o sinal de <b>CTS</b> não é levantado dentro deste <i>time-out</i> , o Driver falha a comunicação atual e retorna erro.
<b>Delay before send</b>	Alguns hardwares de porta serial demoram a habilitar o circuito de envio de dados depois que o sinal <b>RTS</b> é ligado. Configure esta opção para aguardar uma determinada quantidade de milissegundos depois de ligar o sinal <b>RTS</b> e antes de enviar o primeiro byte. <b>IMPORTANTE:</b> Esta espera deve ser utilizada com muito cuidado, pois consome 100% dos recursos da CPU enquanto aguarda. A performance geral do sistema se degrada conforme este valor aumenta.
<b>Delay after send</b>	Tem o mesmo efeito que a opção <b>Delay before send</b> , mas neste caso a espera é efetuada depois que o último byte é enviado, antes de desligar o sinal <b>RTS</b> .

## Aba Ethernet

Utilize esta aba para configurar os parâmetros da Interface **Ethernet**. Estes parâmetros (todos exceto as configurações da porta) devem ser também configurados para uso na Interface **RAS**.

Aba Ethernet

### Opções disponíveis na aba Ethernet

OPÇÃO	DESCRIÇÃO
<b>Transport</b>	Selecione <b>TCP/IP</b> para um <i>socket</i> TCP ( <i>stream</i> ). Selecione <b>UDP/IP</b> para utilizar um <i>socket</i> UDP ( <i>connectionless datagram</i> )
<b>Listen for connections on port</b>	Utilize esta opção para aguardar por novas conexões em uma porta IP específica (comum em Drivers Escravos). Caso esta opção permaneça desmarcada, o Driver se conecta ao endereço e porta especificados na opção <b>Connect to</b>
<b>Share listen port with other processes</b>	Selecione esta opção para compartilhar a porta de escuta com outros Drivers e processos
<b>Interface</b>	Selecione a interface de rede local (identificada por seu endereço IP) que o Driver utiliza para efetuar e receber conexões, ou selecione o item ( <b>All Interfaces</b> ) para permitir conexões em qualquer interface de rede
<b>Use IPv6</b>	Selecione esta opção para forçar o Driver a utilizar endereços no formato <b>IPv6</b> em todas as conexões Ethernet. Deixe esta opção desmarcada para utilizar o formato <b>IPv4</b>
<b>Enable 'ECHO' suppression</b>	Habilite esta opção para eliminar o <i>eco</i> dos dados recebidos. O <i>eco</i> é uma cópia dos dados que foram enviados, que pode ser retornada antes da mensagem de resposta

OPÇÃO	DESCRIÇÃO
<b>IP Filter</b>	Lista de endereços IP restringidos ou permitidos de onde o Driver aceita conexões ( <i>Firewall</i> ). Veja a propriedade <b>IO.Ethernet.IPFilter</b> para mais detalhes
<b>Main IP</b> <b>Backup IP 1</b> <b>Backup IP 2</b> <b>Backup IP 3</b>	<p>Estas opções permitem configurar até quatro endereços para o dispositivo remoto:</p> <ul style="list-style-type: none"> <li>• <b>IP:</b> Digite o endereço IP do dispositivo remoto. Pode-se usar tanto o endereço IP separado por pontos quanto uma URL. No caso de uma URL, o Driver usa o serviço de DNS disponível para mapear a URL para um endereço IP. Por exemplo, "192.168.0.13" ou "Server1"</li> <li>• <b>Port:</b> Digite a porta IP do dispositivo remoto (de 0 até 65535)</li> <li>• <b>Specify local port:</b> Selecione esta opção para utilizar uma porta local fixa ao conectar ao dispositivo remoto</li> </ul>
<b>PING before connecting</b>	<p>Habilite esta opção para executar um comando <b>ping</b> (verificar se o dispositivo pode ser encontrado na rede) no dispositivo antes de tentar uma conexão com o <i>socket</i>. Esta é uma maneira rápida de determinar uma conexão bem sucedida antes de tentar abrir um <i>socket</i> com o dispositivo (o <i>time-out</i> de uma conexão com um <i>socket</i> pode ser bem alto):</p> <ul style="list-style-type: none"> <li>• <b>Timeout:</b> Especifique o número de milissegundos de espera por uma resposta do comando <b>ping</b>. Deve-se usar o comando <b>ping</b> para verificar o tempo normal de resposta, configurando esta opção para um valor acima desta média. Normalmente pode-se configurar um valor entre 1000 e 4000 milissegundos (entre um e quatro segundos)</li> <li>• <b>Retries:</b> Número de retentativas do comando <b>ping</b> (não conta a tentativa inicial). Se todas as tentativas falharem, então a conexão com o <i>socket</i> é abortada</li> </ul>

## Aba Modem

Utilize esta aba para configurar os parâmetros da Interface **Modem**. Algumas opções da aba **Serial** afetam a configuração do modem, portanto é interessante não esquecer de configurar a Interface **Serial**.

**Aba Modem**

A Interface **Modem** utiliza os modems TAPI instalados no computador.

### Opções disponíveis na aba Modem

OPÇÃO	DESCRIÇÃO
<b>Select the modem to use</b>	Selecione um modem a partir da lista de modems disponíveis no computador. Caso selecione-se a opção <b>Default modem</b> , então o primeiro modem disponível é utilizado. Selecionar esta opção é recomendado especialmente quando a aplicação é utilizada em outro computador.
<b>Modem settings</b>	Clique para abrir a janela de configuração do modem selecionado.
<b>Dial Number</b>	Digite o número padrão para discagem (este valor pode ser modificado em tempo de execução). Pode-se utilizar o caractere <b>w</b> para representar uma pausa (espera pelo tom de discagem). Por exemplo, "0w33313456" (disca o número zero, espera e então disca o número "33313456").
<b>Accept incoming calls</b>	Habilite esta opção para que o Driver atenda o telefone quando receber uma chamada externa. Para utilizar esta opção é necessário configurar a opção <b>Connection management</b> na aba <b>Setup</b> para <b>Manual</b> .

## Aba RAS

Use esta aba para configurar os parâmetros da Interface **RAS**. É necessário também configurar a aba **Ethernet**.

A Interface **RAS** abre uma conexão *socket* com um dispositivo RAS. O dispositivo RAS é um servidor de modems acessível através de TCP/IP, aguardando por conexões *socket* em uma porta IP. Para cada conexão aceita nesta porta tem-se acesso a um modem.

Ao conectar-se a um dispositivo RAS, primeiramente a Interface de Comunicação conecta ao *socket* no endereço IP e na porta configurados na aba **Ethernet**. Depois que o *socket* é aberto, os seguintes passos de inicialização ou de conexão são efetuados:

1. Limpeza do *socket* (remove qualquer mensagem de saudação **TELNET** recebida do dispositivo RAS).
2. Envio do comando de discagem **AT** (em ASCII) no *socket*.
3. Aguarda pela recepção de uma resposta **CONNECT**.
4. Caso o *time-out* expire, a conexão é abortada.
5. Se a resposta **CONNECT** é recebida dentro do *time-out*, o *socket* está disponível para comunicação com o dispositivo (a conexão foi estabelecida).

Se o passo 5 é efetuado com sucesso, então o *socket* comporta-se como um *socket* normal, com o dispositivo RAS funcionando como um roteador entre o Driver e o dispositivo. Os bytes enviados pelo Driver são recebidos pelo dispositivo RAS e enviados para o dispositivo de destino utilizando um modem. Os bytes recebidos pelo dispositivo RAS do modem são enviados de volta ao Driver utilizando o mesmo *socket*.

Depois que a conexão é estabelecida, a Interface **RAS** monitora os dados recebidos pelo Driver. Caso uma **String** "NO CARRIER" seja encontrada, o *socket* é fechado. Se o dispositivo RAS não envia o sinal **NO CARRIER**, a Interface **RAS** não consegue detectar quando a conexão modem entre o dispositivo RAS e o dispositivo final de I/O falha. Para recuperação de tal falha é fortemente recomendado que seja habilitada a opção **Disconnect if non-responsive** na aba **Setup**.

The screenshot shows the 'RAS' tab of a configuration window. It contains two input fields: 'AT command:' followed by an empty text box, and 'Connection timeout:' followed by a numeric input box containing '0' and the text 'seconds'. Below these fields, a message states: 'Other socket settings should be configured in the "Ethernet" tab!'. The tab is labeled 'RAS' at the top right.

Aba RAS

### Opções disponíveis na aba RAS

OPÇÃO	DESCRIÇÃO
<b>AT command</b>	Uma <b>String</b> com o comando <b>AT</b> completo usado para discar para o dispositivo de destino. Por exemplo, "ATDT33313456" (discagem por tom para o número "33313456").
<b>Connection timeout</b>	Número de segundos a aguardar por uma resposta <b>CONNECT</b> do modem, após o envio do comando <b>AT</b> .

## Configurações Gerais

Esta seção contém informações sobre a configuração dos **Tags de Comunicação** e as **Propriedades** gerais das Interfaces de Comunicação.

## Tags de Comunicação

### Tags gerais das Interfaces de Comunicação (N2/B2 = 0)

Os Tags descritos a seguir são fornecidos para todas as Interfaces de I/O suportadas.

### IO.IOKitEvent

<b>Tipo de Tag</b>	Tag Bloco
<b>Tipo de Acesso</b>	Somente Leitura
<b>Parâmetro B1</b>	-1
<b>Parâmetro B2</b>	0
<b>Parâmetro B3</b>	0
<b>Parâmetro B4</b>	1
<b>Propriedade Size</b>	4
<b>Propriedade ParamItem</b>	IO.IOKitEvent

Este Bloco retorna eventos de Driver gerados por várias fontes nas Interfaces de Comunicação. A propriedade **TimeStamp** do Bloco representa o momento em que o evento ocorreu. Os Elementos de Bloco são os seguintes:

- **Elemento 0:** Tipo de evento
  - **0:** Informação
  - **1:** Advertência
  - **2:** Erro
- **Elemento 1:** Fonte do evento
  - **0:** Driver (específico do Driver)
  - **-1:** IOKit (eventos genéricos da Interface de Comunicação)
  - **-2:** Interface **Serial**

- -3: Interface **Modem**
- -4: Interface **Ethernet**
- -5: Interface **RAS**
- **Elemento 2:** Número do erro (específico de cada fonte de evento)
- **Elemento 3:** Mensagem do evento (**String**, específica de cada evento)

**NOTA**

O Driver mantém um número máximo de 100 eventos internamente. Se eventos adicionais são reportados, os eventos mais antigos são descartados.

**IO.PhysicalLayerStatus**

<b>Tipo de Tag</b>	Tag de Comunicação
<b>Tipo de Acesso</b>	Somente Leitura
<b>Parâmetro N1</b>	-1
<b>Parâmetro N2</b>	0
<b>Parâmetro N3</b>	0
<b>Parâmetro N4</b>	2
<b>Configuração por String</b>	IO.PhysicalLayerStatus

Este Tag indica o estado da camada física. Seus possíveis valores são os seguintes:

- **0:** Camada física parada (o Driver está em modo **Offline**, a camada física falhou ao inicializar ou excedeu o número máximo de tentativas de reconexão)
- **1:** Camada física iniciada mas não conectada (o Driver está em modo **Online**, mas a camada física não está conectada. Se a opção **Connection management** estiver configurada para **Automatic**, a camada física pode estar conectando, desconectando ou esperando por uma tentativa de reconexão. Se a opção **Connection management** estiver configurada para **Manual**, então a camada física permanece neste estado até ser forçada a conectar)
- **2:** Camada física conectada (a camada física está pronta para ser usada). Isto **NÃO** significa que o equipamento esteja conectado, apenas que a camada de acesso está funcionando

**IO.SetConfigurationParameters**

<b>Tipo de Tag</b>	Tag Bloco
<b>Tipo de Acesso</b>	Somente Leitura
<b>Parâmetro B1</b>	-1
<b>Parâmetro B2</b>	0
<b>Parâmetro B3</b>	0
<b>Parâmetro B4</b>	3
<b>Propriedade Size</b>	2
<b>Propriedade ParamItem</b>	IO.SetConfigurationParameters



Use este Tag para modificar qualquer propriedade da caixa de diálogo de configuração do Driver em tempo de execução.

Este Tag funciona somente enquanto o Driver estiver em modo **Offline**. Para iniciar o Driver em modo **Offline**, selecione a opção **Start driver OFFLINE** na caixa de diálogo de configuração do Driver. Pode-se tanto escrever em um Tag PLC ou em um Tag Bloco contendo os parâmetros a serem modificados (escritas de Elementos de Bloco individuais não são suportadas, o Bloco inteiro precisa ser escrito de uma vez só).

No **Eclipse SCADA** é necessário usar um Tag Bloco. Cada parâmetro a ser configurado utiliza dois Elementos de Bloco. Por exemplo, caso seja necessário configurar três parâmetros, então o tamanho do Bloco deve ser 6 (3 \* 2). O primeiro Elemento é o nome da propriedade (como uma **String**) e o segundo Elemento é o valor da propriedade. Veja este script no **Eclipse SCADA**:

```
// 'Block' deve ser um Tag Block com leitura automática,
// leitura por varredura e escrita automática desabilitadas.
// Configura os parâmetros
Block.element001 = "IO.Type" // Parâmetro 1
Block.element002 = "Serial"
Block.element003 = "IO.Serial.Port" // Parâmetro 2
Block.element004 = 1
Block.element005 = "IO.serial.BaudRate" // Parâmetro 3
Block.element006 = 19200
// Escreve o Block inteiro
Block.Write()
```

Ao usar o **E3**, a habilidade de criar *arrays* em tempo de execução permite o uso tanto de um Tag de Comunicação quanto de um Tag Bloco. Pode-se utilizar o método **Write** do Driver para enviar os parâmetros diretamente para o Driver, sem a necessidade de criar um Tag. Veja estes exemplos:

```
Dim arr(6)
' Configura os elementos do array
arr(1) = "IO.Type"
arr(2) = "Serial"
arr(3) = "IO.Serial.Port"
arr(4) = 1
arr(5) = "IO.serial.BaudRate"
arr(6) = 19200
' Há dois métodos de enviar os parâmetros
' Método 1: Usando um Tag de Comunicação
tag.WriteEx arr
' Método 2: Sem utilizar um Tag
Driver.Write -1, 0, 0, 3, arr
```

Uma variação do exemplo anterior usa um *array* bidimensional:

```
Dim arr(10)
' Configura os elementos do array. Note que o array foi redimensionado
' para 10 elementos. Elementos vazios do array são ignorados pelo Driver.
arr(1) = Array("IO.Type", "Serial")
arr(2) = Array("IO.Serial.Port", 1)
arr(3) = Array("IO.serial.BaudRate", 19200)
Driver.Write -1, 0, 0, 3, arr
```

O Driver não valida nomes de parâmetros ou valores passados, por isto tenha cuidado ao escrever parâmetros e valores. O comando **Write** falha se o *array* de configuração é criado incorretamente. Pode-se consultar o log do Driver ou usar o parâmetro *writeStatus* do método **WriteEx** para descobrir a causa exata do erro:

```
Dim arr(10), strError
arr(1) = Array("IO.Type", "Serial")
arr(2) = Array("IO.Serial.Port", 1)
arr(3) = Array("IO.serial.BaudRate", 19200)
If Not Driver.WriteEx -1, 0, 0, 3, arr, , strError Then
    MsgBox "Falha ao configurar os parâmetros do Driver: " + strError
End If
```

## IO.WorkOnline

<b>Tipo de Tag</b>	Tag de Comunicação
<b>Tipo de Acesso</b>	Leitura ou Escrita
<b>Parâmetro N1</b>	-1
<b>Parâmetro N2</b>	0
<b>Parâmetro N3</b>	0
<b>Parâmetro N4</b>	4
<b>Configuração por String</b>	IO.WorkOnline

Este Tag informa o estado atual do Driver e permite iniciar ou parar a camada física.

- **0 - Driver Offline:** A camada física está fechada (parada). Este modo permite uma configuração dinâmica dos parâmetros do Driver através do Tag **IO.SetConfigurationParameters**
- **1 - Driver Online:** A camada física está aberta (em execução). Enquanto estiver em modo **Online**, a camada física pode ser conectada ou desconectada (seu estado atual pode ser conferido no Tag **IO.PhysicalLayerStatus**)

No exemplo a seguir (utilizando o **E3**), o Driver é colocado em modo **Offline**, sua porta COM é modificada e então é colocado em modo **Online** novamente:

```
' Configura o Driver em modo Offline
Driver.Write -1, 0, 0, 4, 0
' Muda a porta para COM2
Driver.Write -1, 0, 0, 3, Array("IO.Serial.Port", 2)
' Configura o Driver em modo Online
Driver.Write -1, 0, 0, 4, 1
```

O método **Write** pode falhar quando estiver configurando o Driver em modo **Online** (escrevendo o valor um). Neste caso, o Driver permanece em modo **Offline**. A causa da falha pode ser:

- Tipo de camada física configurada incorretamente (provavelmente um valor inválido foi configurado para a propriedade **IO.Type**)
- O Driver pode ter ficado sem memória
- A camada física pode ter deixado de criar seu *thread* de trabalho (procure no arquivo de log pela mensagem "Failed to create physical layer thread!")
- A camada física não conseguiu inicializar. A causa da falha depende do tipo de camada física. Pode ser um número de porta serial inválida, falha ao inicializar o Windows Sockets, falha ao inicializar o TAPI (modem), etc. A causa é gravada no arquivo de log

### IMPORTANTE

Mesmo que a configuração do Driver para o modo **Online** seja bem sucedida, isto não significa necessariamente que a camada física esteja pronta para uso (pronta para executar operações de entrada e saída com um equipamento externo). O Tag **IO.PhysicalLayerStatus** deve ser verificado para assegurar que a camada física esteja conectada e preparada para a comunicação.

## Propriedades

Estas são as propriedades gerais de todas as Interfaces de I/O suportadas.

### IO.ConnectionMode

9 Controla o modo de gerenciamento da Conexão:

- **0**: Modo automático (o Driver gerencia a conexão)
- **1**: Modo manual (a aplicação gerencia a conexão)

### IO.GiveUpEnable

■ Quando configurada para Verdadeiro, define um número máximo de tentativas de reconexão. Se todas as reconexões falharem, o Driver entra em modo **Offline**. Se configurada para Falso, o Driver tenta até que uma reconexão seja bem-sucedida.

### IO.GiveUpTries

9 Número de tentativas de reconexão antes que esta seja abortada. Por exemplo, se o valor desta propriedade é igual a 1 (um), o Driver tenta apenas uma reconexão quando a conexão é perdida. Se esta falhar, o Driver entra em modo **Offline**.

### IO.InactivityEnable

■ Configure em Verdadeiro para habilitar e em Falso para desabilitar a detecção de inatividade. A camada física é desconectada se estiver inativa por um certo período de tempo. A camada física é considerada inativa apenas se é capaz de enviar dados mas não de recebê-los de volta.

### IO.InactivityPeriodSec

9 Número de segundos para a verificação de inatividade. Se a camada física estiver inativa por este período de tempo, ela é desconectada.

### IO.RecoverEnable

■ Configure em Verdadeiro para habilitar o Driver a recuperar conexões perdidas e em Falso para deixar o Driver em modo **Offline** quando uma conexão é perdida.

### IO.RecoverPeriodSec

9 Tempo de espera entre duas tentativas de conexão, em segundos.

#### NOTA

A primeira reconexão é executada imediatamente após a conexão ser perdida.

## IO.StartOffline

☑ Configure em Verdadeiro para iniciar o Driver em modo **Offline** e em Falso para iniciar o Driver em modo **Online**.

### NOTA

Não faz sentido modificar esta propriedade em tempo de execução, já que ela só pode ser modificada quando o Driver já estiver em modo **Offline**. Para configurar o Driver em modo **Online** em tempo de execução, escreva o valor 1 (um) no Tag **IO.WorkOnline**.

## IO.TimeoutMs

9 Define o *time-out* da camada física, em milissegundos (um segundo é igual a 1000 milissegundos).

## IO.Type

A Define o tipo de interface física utilizada pelo Driver. Os valores possíveis são os seguintes:

- **N ou None**: Não utiliza uma interface física (o Driver deve fornecer uma interface personalizada)
- **S ou Serial**: Utiliza uma porta serial local (COMn)
- **M ou Modem**: Utiliza um modem local (interno ou externo) acessado via TAPI (*Telephony Application Programming Interface*)
- **E ou Ethernet**: Utiliza um *socket* TCP/IP ou UDP/IP
- **R ou RAS**: Utiliza uma Interface **RAS** (*Remote Access Server*). O Driver conecta-se ao equipamento RAS através da Interface **Ethernet** e então emite um comando **AT** (*dial*)

## Configuração de Estatísticas

Esta seção contém informações sobre a configuração dos **Tags de Comunicação** e as **Propriedades** das estatísticas das Interfaces de Comunicação.

## Tags de Comunicação

### Tags de estatísticas das Interfaces de Comunicação (N2/B2 = 0)

Os Tags descritos a seguir mostram estatísticas para todas as Interfaces de Comunicação.

### IO.Stats.Partial.BytesRecv

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Leitura
Parâmetro N1	-1
Parâmetro N2	0
Parâmetro N3	0
Parâmetro N4	1101
Configuração por String	IO.Stats.Partial.BytesRecv

Este Tag retorna a quantidade de bytes recebidos na conexão atual.

## IO.Stats.Partial.BytesSent

<b>Tipo de Tag</b>	Tag de Comunicação
<b>Tipo de Acesso</b>	Somente Leitura
<b>Parâmetro N1</b>	-1
<b>Parâmetro N2</b>	0
<b>Parâmetro N3</b>	0
<b>Parâmetro N4</b>	1100
<b>Configuração por String</b>	IO.Stats.Partial.BytesSent

Este Tag retorna a quantidade de bytes enviados na conexão atual.

## IO.Stats.Partial.TimeConnectedSeconds

<b>Tipo de Tag</b>	Tag de Comunicação
<b>Tipo de Acesso</b>	Somente Leitura
<b>Parâmetro N1</b>	-1
<b>Parâmetro N2</b>	0
<b>Parâmetro N3</b>	0
<b>Parâmetro N4</b>	1102
<b>Configuração por String</b>	IO.Stats.Partial.TimeConnectedSeconds

Este Tag retorna o número de segundos que o Driver está conectado na conexão atual ou 0 (zero) se o Driver está desconectado.

## IO.Stats.Partial.TimeDisconnectedSeconds

<b>Tipo de Tag</b>	Tag de Comunicação
<b>Tipo de Acesso</b>	Somente Leitura
<b>Parâmetro N1</b>	-1
<b>Parâmetro N2</b>	0
<b>Parâmetro N3</b>	0
<b>Parâmetro N4</b>	1103
<b>Configuração por String</b>	IO.Stats.Partial.TimeDisconnectedSeconds

Este Tag retorna o número de segundos que o Driver está desconectado desde o término da última conexão ou 0 (zero) se o Driver está conectado.

## IO.Stats.Total.BytesRecv

<b>Tipo de Tag</b>	Tag de Comunicação
<b>Tipo de Acesso</b>	Somente Leitura
<b>Parâmetro N1</b>	-1
<b>Parâmetro N2</b>	0
<b>Parâmetro N3</b>	0
<b>Parâmetro N4</b>	1001
<b>Configuração por String</b>	IO.Stats.Total.BytesRecv

Este Tag retorna a quantidade de bytes recebidos desde que o Driver foi carregado.

## IO.Stats.Total.BytesSent

<b>Tipo de Tag</b>	Tag de Comunicação
<b>Tipo de Acesso</b>	Somente Leitura
<b>Parâmetro N1</b>	-1
<b>Parâmetro N2</b>	0
<b>Parâmetro N3</b>	0
<b>Parâmetro N4</b>	1000
<b>Configuração por String</b>	IO.Stats.Total.BytesSent

Este Tag retorna a quantidade de bytes enviados desde que o Driver foi carregado.

## IO.Stats.Total.ConnectionCount

<b>Tipo de Tag</b>	Tag de Comunicação
<b>Tipo de Acesso</b>	Somente Leitura
<b>Parâmetro N1</b>	-1
<b>Parâmetro N2</b>	0
<b>Parâmetro N3</b>	0
<b>Parâmetro N4</b>	1004
<b>Configuração por String</b>	IO.Stats.Total.ConnectionCount

Este Tag retorna a quantidade de conexões que o Driver já estabeleceu, com sucesso, desde que foi carregado.

## IO.Stats.Total.TimeConnectedSeconds

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Leitura
Parâmetro N1	-1
Parâmetro N2	0
Parâmetro N3	0
Parâmetro N4	1002
Configuração por String	IO.Stats.Total.TimeConnectedSeconds

Este Tag retorna o número de segundos que o Driver permaneceu conectado desde que foi carregado.

## IO.Stats.Total.TimeDisconnectedSeconds

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Leitura
Parâmetro N1	-1
Parâmetro N2	0
Parâmetro N3	0
Parâmetro N4	1003
Configuração por String	IO.Stats.Total.TimeDisconnectedSeconds

Este Tag retorna o número de segundos que o Driver permaneceu desconectado desde que foi carregado.

## Propriedades

Atualmente, não existem propriedades definidas especificamente para mostrar as estatísticas das Interfaces de Comunicação em tempo de execução.

## Configuração da Interface Ethernet

Esta seção contém informações sobre a configuração dos **Tags de Comunicação** e as **Propriedades** da Interface **Ethernet**.

## Tags de Comunicação

### Tags da Interface Ethernet (N2/B2 = 4)

Os Tags descritos a seguir permitem controlar e identificar a Interface **Ethernet** em tempo de execução (também são válidos quando a Interface **RAS** estiver selecionada).

#### IMPORTANTE

Estes Tags estão disponíveis **SOMENTE** enquanto o Driver estiver em modo **Online**.

## IO.Ethernet.IPSelect

<b>Tipo de Tag</b>	Tag de Comunicação
<b>Tipo de Acesso</b>	Leitura ou Escrita
<b>Parâmetro N1</b>	-1 (menos um)
<b>Parâmetro N2</b>	0 (zero)
<b>Parâmetro N3</b>	4
<b>Parâmetro N4</b>	0 (zero)
<b>Configuração por String</b>	IO.Ethernet.IPSelect

Indica o endereço IP ativo. Os valores possíveis são os seguintes:

- **0:** O endereço principal de IP está selecionado
- **1:** O endereço alternativo (*backup*) de IP está selecionado
- **2:** O endereço alternativo (*backup*) 2 de IP está selecionado
- **3:** O endereço alternativo (*backup*) 3 de IP está selecionado

Se a Interface **Ethernet** (ou **RAS**) estiver conectada, este Tag indica qual dos quatro endereços IP configurados está em uso. Se a Interface estiver desconectada, este Tag indica qual endereço IP é usado primeiro na próxima tentativa de conexão.

Durante o processo de conexão, se o endereço IP ativo não estiver disponível, a Interface de Comunicação tenta conectar-se usando o outro endereço IP. Se a conexão com o endereço IP alternativo funcionar, este é configurado como o endereço IP ativo (*switchover* automático).

Para forçar um *switchover* manual, escreva valores de 0 (zero) a três (3) neste Tag. Isto força a reconexão com o endereço IP especificado (**0:** Endereço principal, **1, 2, 3:** Endereços alternativos) se o Driver estiver atualmente conectado. Se o Driver estiver desconectado, este Tag configura o endereço IP ativo para a próxima tentativa de conexão.

## IO.Ethernet.IPSwitch

<b>Tipo de Tag</b>	Tag de Comunicação
<b>Tipo de Acesso</b>	Somente Escrita
<b>Parâmetro N1</b>	-1 (menos um)
<b>Parâmetro N2</b>	0 (zero)
<b>Parâmetro N3</b>	4
<b>Parâmetro N4</b>	1 (um)
<b>Configuração por String</b>	IO.Ethernet.IPSwitch

Qualquer valor escrito neste Tag força um *switchover* manual. Se o endereço principal de IP estiver ativo, então o primeiro endereço alternativo de IP (*backup*) é ativado, e assim por diante passando por todos os endereços alternativos de IP e voltando para o endereço principal até conseguir uma conexão.

Se o Driver estiver desconectado, este Tag configura o endereço IP ativo para a próxima tentativa de conexão.



## Propriedades

Estas propriedades controlam a configuração da Interface **Ethernet**.

### NOTA

A Interface **Ethernet** também é usada pela Interface **RAS**.

## IO.Ethernet.AcceptConnection

■ Configure em Falso se o Driver não deve aceitar conexões externas (o Driver se comporta como mestre) ou configure em Verdadeiro para habilitar a recepção de conexões (o Driver se comporta como escravo).

## IO.Ethernet.BackupEnable

■ Configure em Verdadeiro para habilitar o endereço IP de reserva (*backup*). Se a tentativa de reconectar com o endereço IP principal falhar, o Driver tenta utilizar o endereço IP de reserva. Configure em Falso para desabilitar sua utilização.

## IO.Ethernet.BackupIP

📌 Endereço IP alternativo (*backup*) do equipamento de destino. Pode-se utilizar tanto o endereço numérico como o nome de *host* do equipamento, como por exemplo "192.168.0.7" ou "SERVER2".

## IO.Ethernet.BackupLocalPort

9 Número da porta local a ser utilizada na conexão ao endereço IP alternativo do equipamento de destino. Usado apenas se **IO.Ethernet.BackupLocalPortEnable** for Verdadeiro.

## IO.Ethernet.BackupLocalPortEnable

■ Configure em Verdadeiro para forçar o uso de uma porta local específica ao conectar ao endereço IP de reserva (*backup*). Configure em Falso para utilizar qualquer porta local disponível.

## IO.Ethernet.BackupPort

9 Número da porta do endereço IP alternativo do equipamento de destino (usado juntamente com a propriedade **IO.Ethernet.BackupIP**).

## IO.Ethernet.IPFilter

**A** Lista de endereços IPv4 ou IPv6 separados por vírgula, que define de quais endereços o Driver aceita ou bloqueia conexões. Pode-se utilizar asteriscos (como por exemplo "192.168.\*.\*") ou intervalos (como por exemplo "192.168.0.41-50") em qualquer parte dos endereços IP. Para bloquear um endereço IP ou um intervalos de endereços IP, use o caractere til ("~") no início do endereço. Exemplos:

- **192.168.0.24**: Aceita apenas conexões do endereço IPv4 192.168.0.24
- **192.168.0.41-50**: Aceita conexões dos endereços IPv4 no intervalo entre 192.168.0.41 e 192.168.0.50
- **192.168.0.\***: Aceita conexões dos endereços IPv4 no intervalo entre 192.168.0.0 e 192.168.0.255
- **fe80:3bf:877::\*:\*** (**expande para fe80:03bf:0877:0000:0000:0000:0000:0000**): Aceita conexões de endereços IPv6 no intervalo entre fe80:03bf:0877:0000:0000:0000:0000:0000 e fe80:03bf:0877:0000:0000:0000:ffff:ffff
- **192.168.0.10, 192.168.0.15, 192.168.0.20**: Aceita conexões dos endereços IPv4 192.168.0.10, 192.168.0.15 e 192.168.0.20
- **~192.168.0.95, 192.168.0.\***: Aceita conexões dos endereços IPv4 no intervalo entre 192.168.0.0 e 192.168.0.255, exceto o endereço IPv4 192.168.0.95

Quando o Driver recebe uma tentativa de conexão, a lista de filtros é percorrida sequencialmente da esquerda para a direita, procurando por uma autorização ou bloqueio específico para o endereço IP de onde veio a conexão. Se nenhum elemento da lista corresponder ao endereço IP, a autorização ou bloqueio são ditados pelo último elemento da lista:

- Se o último elemento da lista é uma autorização (como por exemplo "192.168.0.24"), então todos os endereços IP que não forem encontrados na lista são bloqueados
- Se o último elemento da lista é um bloqueio (como por exemplo "~192.168.0.24"), então todos os endereços IP que não forem encontrados na lista são autorizados

Se um endereço IP aparecer em mais de um filtro da lista, o filtro mais à esquerda tem precedência. Por exemplo, no caso de "~192.168.0.95, 192.168.0.\*", o endereço IP 192.168.0.95 se encaixa nas duas regras, mas a regra que vale é a mais à esquerda ("~192.168.0.95", e portanto o endereço IP é bloqueado).

Quando o **IOKit** bloqueia uma conexão, é logada a mensagem "Blocked incoming socket connection from {IP}!".

No caso de conexões UDP em modo escuta em *broadcast*, onde o Driver pode receber pacotes de diferentes endereços IP, o bloqueio ou permissão é realizado a cada pacote recebido. Se um pacote é recebido de um endereço IP bloqueado, é logada a mensagem "Blocked incoming packet from {IP} (discarding {N} bytes)!".

## IO.Ethernet.ListenIP

**A** Endereço IP da interface local de rede por onde o Driver efetua e aceita conexões. Deixe esta propriedade vazia para efetuar e aceitar conexões por qualquer interface local de rede.

## IO.Ethernet.ListenPort

**9** Número da porta IP utilizada pelo Driver para escutar conexões.

## IO.Ethernet.MainIP

**A** Endereço IP do equipamento de destino. Pode-se utilizar tanto o endereço numérico como o nome de *host* do equipamento, como por exemplo "192.168.0.7" ou "SERVER2".

## IO.Ethernet.MainLocalPort

9 Número da porta local a ser utilizada na conexão ao endereço IP principal do equipamento de destino. Este valor é usado apenas se a propriedade **IO.Ethernet.MainLocalPortEnable** é igual a Verdadeiro.

## IO.Ethernet.MainLocalPortEnable

■ Configure em Verdadeiro para forçar o uso de uma porta local específica ao conectar ao endereço IP principal. Configure em Falso para utilizar qualquer porta local disponível.

## IO.Ethernet.MainPort

9 Número da porta IP no equipamento de destino (usado juntamente com a propriedade **IO.Ethernet.MainIP**).

## IO.Ethernet.PingEnable

■ Configure em Verdadeiro para habilitar o envio de um comando **ping** para o endereço IP do equipamento de destino, antes de tentar conectar-se ao *socket*. O *time-out* de conexão do *socket* não pode ser controlado, por isto o envio de um comando **ping** antes de conectar-se é uma maneira rápida de detectar se a conexão vai falhar. Configure em Falso para desabilitar o comando **ping**.

## IO.Ethernet.PingTimeoutMs

9 Tempo de espera por uma resposta de um comando **ping**, em milissegundos.

## IO.Ethernet.PingTries

9 Número máximo de tentativas de comandos **ping**. O valor mínimo é 1 (um), incluindo o primeiro comando **ping**.

## IO.Ethernet.ShareListenPort

■ Configure em Verdadeiro para compartilhar a porta de escuta com outros Drivers e processos ou Falso para abrir a porta de escuta em modo exclusivo. Para compartilhar uma porta de escuta com sucesso, todos os Drivers e processos envolvidos devem abrir esta porta em modo compartilhado. Quando uma porta de escuta é compartilhada, cada nova conexão é distribuída para um dos processos que estão escutando. Desta forma, se um Driver Escravo só suporta uma conexão por vez, pode-se utilizar várias instâncias deste Driver escutando na mesma porta, portanto simulando um Driver com suporte a múltiplas conexões.

## IO.Ethernet.SuppressEcho

■ Configure em Verdadeiro para eliminar o eco presente em uma comunicação. O eco é a recepção indesejada de uma cópia exata de todos os pacotes de dados que o Driver enviou para o equipamento.

## IO.Ethernet.Transport

A Define o protocolo de transporte. Os valores possíveis são os seguintes:

- **T ou TCP**: Utiliza o protocolo TCP/IP
- **U ou UDP**: Utiliza o protocolo UDP/IP

## IO.Ethernet.UseIPv6

■ Configure em Verdadeiro para utilizar endereços IPv6 em todas as conexões Ethernet. Configure em Falso para utilizar endereços IPv4 (este é valor padrão).

# Configuração da Interface Modem

Esta seção contém informações sobre a configuração dos **Tags de Comunicação** e as **Propriedades** da Interface **Modem** (TAPI).

## Tags de Comunicação

### Tags da Interface Modem (N2/B2 = 3)

Os Tags descritos a seguir permitem controlar e diagnosticar a Interface **Modem** (TAPI) em tempo de execução.

#### IMPORTANTE

Estes Tags estão disponíveis **SOMENTE** enquanto o Driver estiver em modo **Online**.

### IO.TAPI.ConnectionBaudRate

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Leitura
Parâmetro N1	-1
Parâmetro N2	0
Parâmetro N3	3
Parâmetro N4	5
Configuração por String	IO.TAPI.ConnectionBaudRate

Indica o valor de *baud rate* da conexão atual. Se o modem não estiver conectado, retorna o valor 0 (zero).

### IO.TAPI.Dial

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Escrita
Parâmetro N1	-1
Parâmetro N2	0
Parâmetro N3	3
Parâmetro N4	1
Configuração por String	IO.TAPI.Dial

Escreva qualquer valor neste Tag para forçar a Interface **Modem** a iniciar uma chamada. Este comando é assíncrono, apenas iniciando o processo de chamada. Pode-se monitorar o Tag **IO.TAPI.IsModemConnected** para detectar quando a chamada é estabelecida.

## IO.TAPI.ModemStatus

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Leitura
Parâmetro N1	-1
Parâmetro N2	0
Parâmetro N3	3
Parâmetro N4	2
Configuração por String	IO.TAPI.ModemStatus

Retorna uma **String** com o estado atual do modem. Os valores possíveis são os seguintes:

- **"No status!"**: A Interface **Modem** ainda não foi aberta ou já foi fechada
- **"Modem initialized OK!"**: A Interface **Modem** foi inicializada com sucesso
- **"Modem error at initialization!"**: O Driver não conseguiu inicializar a linha do modem. Confira o arquivo de log do Driver para maiores detalhes
- **"Modem error at dial!"**: O Driver não conseguiu começar ou aceitar uma chamada
- **"Connecting..."**: O Driver iniciou uma chamada com sucesso, e está atualmente processando esta chamada
- **"Ringing..."**: Indica que o modem está recebendo uma chamada externa, mas ainda não a aceitou
- **"Connected!"**: O Driver conectou-se com sucesso (completou ou aceitou uma chamada externa)
- **"Disconnecting..."**: O Driver está desligando a chamada atual
- **"Disconnected OK!"**: O Driver desligou a chamada atual
- **"Error: no dial tone!"**: O Driver abortou a chamada porque o sinal de linha disponível não foi detectado
- **"Error: busy!"**: O Driver abortou a ligação porque a linha estava ocupada
- **"Error: no answer!"**: O Driver abortou a chamada porque não recebeu resposta do outro modem
- **"Error: unknown!"**: A chamada atual foi abortada por um erro desconhecido

## IO.TAPI.PhoneNumber

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Leitura ou Escrita
Parâmetro N1	-1
Parâmetro N2	0
Parâmetro N3	3
Parâmetro N4	0
Configuração por String	IO.TAPI.PhoneNumber

Este Tag é uma **String** que lê ou modifica o número do telefone utilizado pelo Tag **IO.TAPI.Dial**. Ao modificar este Tag, o novo valor é usado apenas no próximo comando **Dial**.

## IO.TAPI.HangUp

<b>Tipo de Tag</b>	Tag de Comunicação
<b>Tipo de Acesso</b>	Somente Escrita
<b>Parâmetro N1</b>	-1
<b>Parâmetro N2</b>	0
<b>Parâmetro N3</b>	3
<b>Parâmetro N4</b>	4
<b>Configuração por String</b>	IO.TAPI.HangUp

Qualquer valor escrito neste Tag desliga a chamada atual.

### NOTA

Use este comando apenas quando estiver gerenciando a camada física manualmente, ou se estiver explicitamente tentando forçar o Driver a reiniciar a comunicação. Se a camada física estiver configurada para reconexão automática, o Driver imediatamente tenta restabelecer a conexão.

## IO.TAPI.IsModemConnected

<b>Tipo de Tag</b>	Tag de Comunicação
<b>Tipo de Acesso</b>	Somente Leitura
<b>Parâmetro N1</b>	-1
<b>Parâmetro N2</b>	0
<b>Parâmetro N3</b>	3
<b>Parâmetro N4</b>	3
<b>Configuração por String</b>	IO.TAPI.IsModemConnected

Este Tag indica o estado da conexão do modem. Os valores possíveis são os seguintes:

- **0:** O modem não está conectado, mas pode estar realizando ou recebendo uma chamada externa
- **1:** O modem está conectado e o Driver completou ou recebeu uma chamada externa com sucesso. Enquanto estiver neste estado, a camada física consegue enviar ou receber dados

## IO.TAPI.IsModemConnecting

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Leitura
Parâmetro N1	-1
Parâmetro N2	0
Parâmetro N3	3
Parâmetro N4	6
Configuração por String	IO.TAPI.IsModemConnecting

Este Tag indica o estado de conexão do modem, com mais detalhes do que o Tag **IO.TAPI.IsModemConnected**. Os valores possíveis são os seguintes:

- **0**: O modem não está conectado
- **1**: O modem está conectando (realizando ou recebendo uma chamada externa)
- **2**: O modem está conectado. Enquanto estiver neste estado, a camada física consegue enviar ou receber dados
- **3**: O modem está desconectando a chamada atual

## Propriedades

Estas propriedades controlam a configuração da Interface **Modem** (TAPI).

## IO.TAPI.AcceptIncoming

**9** Configure em Falso se o modem não pode aceitar chamadas externas (o Driver se comporta como mestre) e configure em Verdadeiro para habilitar a recepção de chamadas (o Driver se comporta como escravo).

## IO.TAPI.ModemID

**9** É o número de identificação do modem. Este ID é criado pelo Windows e é usado internamente para identificar o modem dentro de uma lista de equipamentos instalados no computador. Este ID pode não permanecer válido caso o modem seja reinstalado ou a aplicação seja executada em outro computador.

### NOTA

Recomenda-se que esta propriedade seja configurada em 0 (zero), indicando que o Driver deve utilizar o primeiro modem disponível.

## IO.TAPI.PhoneNumber

**A** O número de telefone utilizado em comandos **Dial**. Por exemplo, "0w01234566" (o caractere "w" força o modem a esperar por um sinal de chamada).

## Configuração da Interface RAS

Esta seção contém informações sobre a configuração dos **Tags de Comunicação** e as **Propriedades** da Interface **RAS**.

## Tags de Comunicação

### Tags da Interface RAS (N2/B2 = 5)

Atualmente, não existem Tags definidos especificamente para gerenciar a Interface **RAS** em tempo de execução.


## Propriedades

Estas propriedades controlam a configuração da Interface **RAS**.

### NOTA

A Interface **RAS** utiliza a Interface **Ethernet**, que por este motivo também deve ser configurada.

### IO.RAS.ATCommand

 Comando **AT** a ser enviado através do *socket* para forçar o equipamento RAS a realizar uma ligação usando o canal RAS atual. Exemplo: "ATDT6265545".

### IO.RAS.CommandTimeoutSec

 Tempo de espera pela mensagem **CONNECT** em resposta ao comando **AT**, em segundos.

## Configuração da Interface Serial

Esta seção contém informações sobre a configuração dos **Tags de Comunicação** e as **Propriedades** da Interface **Serial**.

## Tags de Comunicação

### Tags da Interface Serial (N2/B2 = 2)

Atualmente, não existem Tags definidos especificamente para gerenciar a Interface **Serial** em tempo de execução.


## Propriedades

Estas propriedades controlam a configuração da Interface **Serial**.

### IO.Serial.Baudrate

 Especifica a taxa de *bauds* da porta serial, como por exemplo 9600.

### IO.Serial.CTSTimeoutMs

 Tempo de espera pelo sinal **CTS**, em milissegundos. Após o sinal **RTS** ser ligado (**ON**), um temporizador é iniciado para esperar pelo sinal **CTS**. Se este temporizador expirar, o Driver aborta o envio de bytes através da porta serial. Disponível apenas quando a propriedade **IO.Serial.RTS** está configurada em **Toggle** e a propriedade **IO.Serial.WaitCTS** está configurada em Verdadeiro.



## IO.Serial.DataBits

**9** Especifica o número de bits de dados para a configuração da porta serial. Os valores possíveis são os seguintes:

- **5:** Cinco bits de dados
- **6:** Seis bits de dados
- **7:** Sete bits de dados
- **8:** Oito bits de dados

## IO.Serial.DelayAfterMs

**9** Número de milissegundos de atraso após o último byte ter sido enviado através da porta serial, mas antes de desligar (**OFF**) o sinal **RTS**. Disponível apenas quando a propriedade **IO.Serial.RTS** está configurada em **Toggle** e a propriedade **IO.Serial.WaitCTS** está configurada em Falso.

## IO.Serial.DelayBeforeMs

**9** Número de milissegundos de atraso após o sinal **RTS** ter sido ligado (**ON**), mas antes dos dados serem enviados. Disponível apenas quando a propriedade **IO.Serial.RTS** está configurada em **Toggle** e a propriedade **IO.Serial.WaitCTS** está configurada em Falso.

## IO.Serial.DTR

**A** Indica o modo como o Driver lida com o sinal **DTR**:

- **OFF:** Sinal **DTR** sempre desligado
- **ON:** Sinal **DTR** sempre ligado

## IO.Serial.InterbyteDelayUs

**9** Tempo de espera, em milissegundos (1/1000000 de um segundo), para cada dois bytes enviados pela Interface **Serial**.

## IO.Serial.InterframeDelayMs

**9** Tempo de espera, em milissegundos, antes de enviar um pacote após o último pacote enviado ou recebido.

## IO.Serial.Parity

**A** Especifica a paridade para a configuração da porta serial. Os valores possíveis são os seguintes:

- **E ou Even:** Paridade par
- **N ou None:** Sem paridade
- **O ou Odd:** Paridade ímpar
- **M ou Mark:** Paridade de marca
- **S ou Space:** Paridade de espaço

## IO.Serial.Port

9 Número da porta serial local:

- **1:** Utiliza a porta COM1
- **2:** Utiliza a porta COM2
- **3:** Utiliza a porta COM3
- **n:** Utiliza a porta COMn

## IO.Serial.RTS

A Indica como o Driver lida com o sinal **RTS**:

- **OFF:** Sinal **RTS** sempre desligado
- **ON:** Sinal **RTS** sempre ligado
- **Toggle:** Liga (**ON**) o sinal **RTS** quando estiver transmitindo dados e desliga (**OFF**) o sinal **RTS** quando não estiver transmitindo dados

## IO.Serial.StopBits

9 Especifica o número de bits de parada para a configuração da porta serial. Os valores possíveis são os seguintes:

- **1:** Um bit de parada
- **2:** Um bit e meio de parada
- **3:** Dois bits de parada

## IO.Serial.SuppressEcho

9 Utilize um valor diferente de 0 (zero) para habilitar a supressão de eco ou 0 (zero) para desabilitá-la.

## IO.Serial.WaitCTS

▣ Configure em Verdadeiro para forçar o Driver a esperar pelo sinal **CTS** antes de enviar bytes quando o sinal **RTS** estiver ligado (**ON**). Disponível apenas quando a propriedade **IO.Serial.RTS** está configurada em **Toggle**.

## Histórico de Revisões do Driver

VERSÃO	DATA	AUTOR	COMENTÁRIOS
4.0.4	22/06/2020	C. Mello	<ul style="list-style-type: none"> <li>• Adicionado um tipo de dados Inteiro de 52 bits via conversão para um tipo de dados Double de 64 bits (Case 21840).</li> </ul>
4.0.3	17/03/2020	C. Mello	<ul style="list-style-type: none"> <li>• Corrigido um problema de configuração <i>offline</i> para as <b>Strings</b> <i>ModiconModbus.ConfigFile</i> e <i>ModiconModbus.UserTypesConfigFile</i> (Case 28566).</li> </ul>

VERSÃO	DATA	AUTOR	COMENTÁRIOS
4.0.2	05/08/2019	M. Ludwig	<ul style="list-style-type: none"> <li>• Driver portado para o Visual Studio 2017 (Case 27087).</li> </ul>
4.0.1	13/11/2018	C. Mello	<ul style="list-style-type: none"> <li>• Corrigido um problema com a configuração <b>1000</b>, que vincula um Tag com o <b>Default Slave Address</b> da janela de propriedades (Case 20713).</li> <li>• Implementado o retorno para tipos de dados <b>BYTE</b> para os valores de registradores Coils (Case 20927).</li> <li>• Corrigido um problema com as escritas de tipos de dados <b>String</b> com tamanho ímpar e com a opção <b>Swap bytes / words / dwords</b> habilitada (Case 23694).</li> <li>• Ajustes de consistência nas edições de estruturas de tipos de dados definidos pelo usuário (Case 20454).</li> </ul>
3.1.36	17/03/2017	C. Mello	<ul style="list-style-type: none"> <li>• Adicionado suporte para inteiros com sinal de magnitude (Case 22091).</li> </ul>
3.1.31	10/07/2016	F. Englert	<ul style="list-style-type: none"> <li>• Resolvido um GPF que poderia ocorrer eventualmente na leitura do histórico de erros do Driver (<math>B2 = 9998</math>) caso ocorresse ao mesmo tempo algum erro de comunicação (Case 21637).</li> </ul>
3.1.30	01/04/2016	C. Mello	<ul style="list-style-type: none"> <li>• Corrigido a falta de valores das Tags configurados por <b>Strings</b> quando o serviço de Superblocos está ativado (Case 20754).</li> </ul>
3.1.28	28/01/2016	A. Quites	<ul style="list-style-type: none"> <li>• Implementado novo Tag Browser com <i>templates</i> de Tags configurados por <b>Strings</b> (Case 20460).</li> </ul>

VERSÃO	DATA	AUTOR	COMENTÁRIOS
3.1.27	27/01/2016	A. Quites	<ul style="list-style-type: none"> <li>Implementada uma opção de endereçamento de Tags por <b>Strings</b> nos parâmetros <b>Item</b> e <b>Device</b> (<i>Case 19119</i>).</li> <li>Corrigido um erro na leitura de Tags de tempo real de eventos do equipamento GE PAC RX7 por <i>callbacks</i> (<i>Case 20374</i>).</li> </ul>
3.1.26	20/01/2016	A. Quites	<ul style="list-style-type: none"> <li>Corrigido um erro pelo qual a rotina <b>Elipse SOE</b> não retornava valores reportados a eventos para tipos de dados nativos, não estruturados (<i>Case 20364</i>).</li> </ul>
3.1.25	19/01/2016	A. Quites	<ul style="list-style-type: none"> <li>Implementada a configuração por <b>Strings</b> para tipos de dados do usuário (<i>Case 19807</i>).</li> </ul>
3.1.24	18/01/2016	A. Quites	<ul style="list-style-type: none"> <li>Adicionada uma verificação para proibir a definição de tipos de dados do usuário com nomes que conflitem com tipos de dados nativos do Driver (<i>Case 19816</i>).</li> <li>Corrigido um erro na leitura de tipos de dados <b>UTC32</b> em Bloco, que retorna valores errados a partir do segundo Elemento do Bloco (<i>Case 19819</i>).</li> </ul>
3.1.23	14/01/2016	A. Quites	<ul style="list-style-type: none"> <li>Adicionada uma validação para evitar erro do usuário ao atribuir tipos de dados de data e hora a elementos da estrutura de tipos de dados nativos (<i>Case 20415</i>).</li> </ul>
3.1.15	22/12/2015	A. Quites	<ul style="list-style-type: none"> <li>Corrigida a escrita em bloco de Tags Bloco com tipo de dados <b>Double</b> (<i>Case 20053</i>).</li> </ul>

VERSÃO	DATA	AUTOR	COMENTÁRIOS
3.1.13	02/12/2015	A. Qites	<ul style="list-style-type: none"> <li>• Corrigido um erro na exportação de arquivos INI em formato compatível com a versão 1.0 do Driver, com os tipos de dados sendo especificados por números e não por <b>Strings</b>, como passou a ocorrer a partir da versão 2.8 (<i>Case 20203</i>).</li> <li>• Corrigido um erro de <i>byte order</i> na escrita de tipos de dados <b>BCD</b> de oito dígitos e tamanho de quatro bytes (<i>Case 20204</i>).</li> </ul>
3.1.9	01/10/2015	A. Qites	<ul style="list-style-type: none"> <li>• Habilitado o armazenamento de últimas exceções para Tags de comunicação configurados por <b>Strings</b> (<i>Case 19808</i>).</li> </ul>
3.1.8	28/09/2016	A. Qites	<ul style="list-style-type: none"> <li>• Corrigido um erro na leitura e escrita de tipos de dados <b>BCD</b> de oito dígitos e quatro bytes superiores a 9999999, que provocavam a finalização do processo (<i>Case 19733</i>).</li> </ul>

VERSÃO	DATA	AUTOR	COMENTÁRIOS
3.0.11	29/05/2015	A. Quites	<ul style="list-style-type: none"> <li>• Driver alterado para evitar possível comportamento estranho na reconexão após <i>time-out</i> em Tags de <i>callback</i> (SOE), com a opção <b>Reconnect after Timeout (Ethernet only)</b> habilitada. Podem ocorrer, por exemplo, duas conexões simultâneas seguidas de duas desconexões (<i>Case 14775</i>).</li> <li>• Corrigido um erro na gravação de operações na janela de configuração do Driver, na aba <b>Operações</b>, que fazia com que ao remover operações no final da lista, eventualmente voltassem à lista quando a janela era fechada clicando em <b>OK</b> e reaberta logo em seguida (<i>Case 14874</i>).</li> <li>• Corrigido um problema que poderia gerar a perda de informações de última exceção em casos raros em que são recebidos <i>frames</i> válidos com CRC correto, porém que não se destinavam à requisição atual. Pode ocorrer quando se utiliza <b>Modbus RTU</b> em meio <b>Ethernet TCP/IP</b> em redes lentas (<i>Case 15314</i>).</li> <li>• Solucionado um vazamento de <i>handle</i> do <i>thread</i> de <i>download</i> de memória de massa do GE PAC RX7 (<i>Case 16404</i>).</li> <li>• Corrigido um erro em que o Driver poderia ignorar os últimos caracteres de <b>Strings</b> lidas do equipamento quando alguma das opções de <i>swap</i> estavam configuradas na operação (<i>Case 16744</i>).</li> </ul>

VERSÃO	DATA	AUTOR	COMENTÁRIOS
			<ul style="list-style-type: none"> <li>• Corrigido um erro na leitura de operações com a opção <b>Use bit mask</b> habilitada quando usada com a propriedade <b>EnableReadGrouping</b> do Driver habilitada (<i>Case 18340</i>).</li> <li>• Resolvido um erro de validação em Tags especiais do Driver (<i>Case 16433</i>).</li> </ul>
<b>3.0.8</b>	31/07/2014	A. Quites	<ul style="list-style-type: none"> <li>• Corrigido um erro que poderia gerar GPFs ou comportamentos inesperados ao executar múltiplas instâncias do Driver em um mesmo I/O Server, sobretudo se estas instâncias possuísem configurações diferentes nas abas <b>Modbus</b> e <b>Operations</b> da janela de configurações do Driver (<i>Case 14856</i>).</li> </ul>
<b>3.0.6</b>	12/06/2014	A. Quites	<ul style="list-style-type: none"> <li>• Driver portado para o <b>IOKit 2.0</b> (<i>Case 13891</i>).</li> </ul>

VERSÃO	DATA	AUTOR	COMENTÁRIOS
2.8.17	19/10/2012	A. Quites	<ul style="list-style-type: none"> <li>Adicionados os tipos de dados ou estruturas definidos pelo usuário como parte da implementação do recurso de SOE Genérico (<i>Case 12038</i>).</li> <li>Implementada a leitura de SOE e comandos para a sincronização do relógio para relés Schneider Electric séries SEPAM 20, 40 e 80 (<i>Case 12106</i>).</li> <li>A opção <b>Reverse Frame</b> foi removida da caixa de diálogo de configuração das operações do Driver, por obsolescência. A opção continua sendo suportada em aplicações legadas como configuração <i>offline</i> apenas (<i>Case 12443</i>).</li> <li>Adicionada a leitura por <i>callbacks</i> para Tags de leitura de SOE (<i>Case 12464</i>).</li> <li>Adicionada uma opção de reconexão em caso de ocorrência de <i>time-out</i> na recepção de <i>frames</i> ao usar a camada física Ethernet (<i>Case 12537</i>).</li> <li>A opção <b>Swap Address Delay</b> foi removida da janela de configuração do Driver. Esta opção, tornada obsoleta pela opção <b>Inter-frame delay</b> do <b>IOMKit</b>, continua disponível como configuração <i>offline</i>, mantendo compatibilidade com aplicações legadas (<i>Case 13285</i>).</li> <li>Solucionado um erro na escrita de valores <b>float_GE</b> (<i>Case 12298</i>).</li> <li>Solucionado um erro ao carregar arquivo de configuração no Windows CE ARM HPC2000 (<i>Case 12352</i>).</li> </ul>



VERSÃO	DATA	AUTOR	COMENTÁRIOS
			<ul style="list-style-type: none"> <li>Solucionado um erro em que a leitura de <b>Strings</b> de tamanho ímpar poderia vir sem o último caractere (<i>Case 12466</i>).</li> </ul>
2.7.1	30/06/2010	A. Quites	<ul style="list-style-type: none"> <li>Implementado o recurso de tornar o Driver imune a ruídos antes de receber o <i>frame</i> em modo <b>RTU</b> (<i>Case 11394</i>).</li> </ul>
2.6.1	26/11/2009	A. Quites	<ul style="list-style-type: none"> <li>Driver portado para o <b>Windows CE</b> (<i>Case 10914</i>).</li> <li>Driver reporta um falso erro no log ao ler um endereço de registro em zero (<i>Case 10654</i>).</li> <li>Otimizada a leitura de bits usando o agrupamento de leituras ou Superblocos (<i>Case 10971</i>).</li> </ul>
		C. Mello	<ul style="list-style-type: none"> <li>A opção <b>Wait Silence</b> não funciona para todos os erros (<i>Case 10850</i>).</li> </ul>
2.5.1	30/06/2009	A. Quites	<ul style="list-style-type: none"> <li>Atualizado o tamanho máximo do PDU para o Driver Modbus (<i>Case 10274</i>).</li> <li>Corrigido um erro ao ler a função Modbus <b>20, Read File Record</b> (<i>Case 10312</i>).</li> <li>Corrigido um erro no Tag Especial para retorno da última exceção. O Tag poderia não reportar algumas exceções (<i>Case 10337</i>).</li> <li>Eventos <b>GE SOE</b> com o dia atual retornavam o ano errado (<i>Case 10382</i>).</li> <li>Endereço de <i>swap</i> não funcionava (<i>Case 10425</i>).</li> </ul>
		M. Ludwig	<ul style="list-style-type: none"> <li>Criado um novo Tag para receber uma lista de eventos de pontos específicos no <b>GE SOE</b> (<i>Case 10370</i>).</li> </ul>

VERSÃO	DATA	AUTOR	COMENTÁRIOS
2.4.1	17/02/2009	A. Quites	<ul style="list-style-type: none"> <li>Adicionadas funções de leitura e escrita por padrão (<i>Case 9185</i>).</li> <li>Driver Modbus com Eventos <b>GE</b>, leitura de último evento por <i>polling</i> (<i>Case 10178</i>).</li> <li>Erro ao ler blocos de bits com agrupamentos de leituras ou Superblocos habilitados (<i>Case 10100</i>).</li> </ul>
2.3.1	02/09/2008	A. Quites	<ul style="list-style-type: none"> <li>Implementado o recurso <b>CMS Extended Device Addressing</b> (<i>Case 8665</i>).</li> <li>Implementada a configuração de tamanho máximo dos Superblocos (<i>Case 9154</i>).</li> <li>O Driver aceita enviar um <b>DWord</b> de 32 bits usando a função Modbus <b>6</b> (<i>Case 8663</i>).</li> <li>Revisão de funções não padrão ou pouco utilizadas: <b>07, 20, 21, 65 01, 65 02 e 65 03</b> (<i>Case 8730</i>).</li> <li>A opção <b>Swap byte</b> não funciona corretamente com Superblocos (<i>Case 9220</i>).</li> <li>Configuração em modo <i>offline</i> do parâmetro <i>ModiconModbus.ModbusMod</i> e (<i>Case 9831</i>).</li> </ul>
2.2.1	11/05/2007	A. Quites	<ul style="list-style-type: none"> <li>Usar o tipo de dados <b>Int16</b> para a leitura de blocos retorna somente o primeiro elemento (<i>Case 8243</i>).</li> </ul>
2.1.1	23/01/2007	A. Quites	<ul style="list-style-type: none"> <li>Implementados os Superblocos (<i>Case 6185</i>).</li> <li>O Mestre Modicon Modbus não testa corretamente a consistência do parâmetro <i>N2/B2</i> (<i>Case 7714</i>).</li> <li>Erro de <i>offset</i> ao ler blocos de tipos de dados <b>BCD</b> com o tamanho do tipo de dados igual a 4 (<i>Case 7728</i>).</li> <li>Leitura de blocos de <b>Strings</b> não estava funcionando corretamente (<i>Case 7804</i>).</li> </ul>

VERSÃO	DATA	AUTOR	COMENTÁRIOS
2.0.1	14/09/2005	A. Quites	<ul style="list-style-type: none"> <li>• Driver portado para o <b>IOKit</b> (<i>Case 2050</i>).</li> </ul>
1.3.1	19/12/2006	C. Mello	<ul style="list-style-type: none"> <li>• Driver comunicando com mais de um endereço IP muda os valores lidos (<i>Case 7191</i>).</li> <li>• Possibilidade de definir uma porta TCP local (<i>Case 7109</i>).</li> <li>• Corrigido um erro de <i>offset</i> ao ler blocos de tipos de dados <b>BCD</b> com o tamanho do tipo de dados igual a 4 (<i>Case 7729</i>).</li> <li>• Driver não testava a consistência dos parâmetros <i>N2/B2</i> corretamente (<i>Case 7735</i>).</li> </ul>
1.2.1	15/12/2005	C. Mello	<ul style="list-style-type: none"> <li>• Ajustes para comunicação com Corretores de Vazão da ICP (<i>Case 4979</i>).</li> <li>• Corrigido um vazamento de <i>handles</i> se o comando <b>ping</b> falhasse (<i>Case 6497</i>).</li> </ul>
1.1.1	22/12/2004	C. Mello	<ul style="list-style-type: none"> <li>• Adicionadas retentativas ao Driver (<i>Case 3365</i>).</li> <li>• Adicionada a função <b>Broadcast</b> (<i>Case 4045</i>).</li> <li>• Adicionada a opção <b>HALT</b> ao Driver (<i>Case 4429</i>).</li> <li>• Corrigido um erro na leitura e escrita de <b>Strings</b> (<i>Case 4386</i>).</li> <li>• Corrigidas as configurações incompatíveis com versões anteriores (<i>Case 4431</i>).</li> </ul>

#### **Matriz**

**Rua 24 de Outubro, 353 - 10º andar**  
**90510-002 Porto Alegre**  
**Fone: (+55 51) 3346-4699**  
**Fax: (+55 51) 3222-6226**  
**E-mail: [elipse-rs@elipse.com.br](mailto:elipse-rs@elipse.com.br)**

#### **Filial SP**

**Rua dos Pinheiros, 870 - Conj. 141/142**  
**05422-001 São Paulo - SP**  
**Fone: (+55 11) 3061-2828**  
**Fax: (+55 11) 3086-2338**  
**E-mail: [elipse-sp@elipse.com.br](mailto:elipse-sp@elipse.com.br)**

#### **Filial PR**

**Av. Sete de Setembro, 4698/1705**  
**80240-000 Curitiba - PR**  
**Fone: (+55 41) 4062-5824**  
**E-mail: [elipse-pr@elipse.com.br](mailto:elipse-pr@elipse.com.br)**

#### **Filial MG**

**Rua Antônio de Albuquerque, 156**  
**7º andar Sala 705**  
**30112-010 Belo Horizonte - MG**  
**Fone: (+55 31) 4062-5824**  
**E-mail: [elipse-mg@elipse.com.br](mailto:elipse-mg@elipse.com.br)**

#### **Filial RJ**

**Praia de Botafogo, 300/525**  
**22250-044 Rio de Janeiro - RJ**  
**Fone: (+55 21) 2158-1015**  
**Fax: (+55 21) 2158-1099**  
**E-mail: [elipse-rj@elipse.com.br](mailto:elipse-rj@elipse.com.br)**

#### **Taiwan**

**9F., No.12, Beiping 2nd St., Sanmin Dist.**  
**807 Kaohsiung City - Taiwan**  
**Fone: (+886 7) 323-8468**  
**Fax: (+886 7) 323-9656**  
**E-mail: [evan@elipse.com.br](mailto:evan@elipse.com.br)**

**Consulte nosso website para informações sobre o representante do seu estado.**

**[www.elipse.com.br](http://www.elipse.com.br)**

**[kb.elipse.com.br](http://kb.elipse.com.br)**

**[forum.elipse.com.br](http://forum.elipse.com.br)**

**[www.youtube.com/elipsesoftware](http://www.youtube.com/elipsesoftware)**

**[elipse@elipse.com.br](mailto:elipse@elipse.com.br)**



Gartner, Cool Vendors in Brazil 2014, April 2014.

Gartner does not endorse any vendor, product or service depicted in its research publications, and does not advise technology users to select only those vendors with the highest ratings. Gartner research publications consist of the opinions of Gartner's research organization and should not be construed as statements of fact. Gartner disclaims all warranties, expressed or implied, with respect to this research, including any warranties of merchantability of fitness for a particular purpose.

**Microsoft Partner**  
Gold Independent Software Vendor (ISV)