Driver Modicon Modbus

Nome do Arquivo	Modbus.dll
Fabricante	Modicon Schneider Electric
Equipamentos	Qualquer equipamento compatível com o protocolo Modbus versão 1.1b
Protocolo	Modbus versão 1.1b
Versão	4.0.18
Última Atualização	01/10/2026
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. Caso haja necessidade de comunicar com dispositivos **Mestres**, utilize o Driver Modbus Slave da **Elipse Software**, que pode ser obtido no *site da empresa*.

Este Driver, a partir da versão **2.0**, passou a ser desenvolvido com a biblioteca **IOKit** da **Elipse Software**. Esta biblioteca é responsável por implementar o acesso ao meio físico, ou seja, **Serial**, **Ethernet**, **Modem** ou **RAS**. Para mais informações sobre a configuração da biblioteca **IOKit**, consulte o tópico **Documentação das Interfaces de Comunicação**.

Recomenda-se iniciar a leitura pelo tópico **Guia de Configuração Rápida** caso um equipamento seja perfeitamente aderente ao protocolo Modbus padrão, definido pela *Organização Modbus (modbus.org)*, e no caso de apenas ler ou escrever bits e registros, sem necessidade dos recursos mais avançados deste Driver.

Para um entendimento mais aprofundado de todas as funcionalidades deste Driver, recomenda-se iniciar lendo, nesta ordem, os capítulos **Adicionando este Driver em uma Aplicação da Elipse Software** 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 tópicos **O Protocolo Modbus**, **Sites Recomendados**, **Funções Suportadas** e **Funções Especiais**.

Guia de Configuração Rápida

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

Se um equipamento é perfeitamente aderente ao protocolo padrão, e se há necessidade de apenas ler ou escrever registros ou bits, os tópicos **Inserindo este Driver**, **Configurando este Driver** e **Configurando Tags de Comunicação** são provavelmente suficientes para a configuração deste Driver.

Inserindo este Driver

No caso de utilizar o **Elipse E3**, o **Elipse Power** ou o **Elipse Water**, consulte o tópico **Adicionando este Driver em uma Aplicação da Elipse Software - Elipse E3, Elipse Power ou Elipse Water**.

No caso de utilizar o **Elipse SCADA**, consulte o tópico **Adicionando este Driver em uma Aplicação da Elipse Software - Elipse SCADA**.

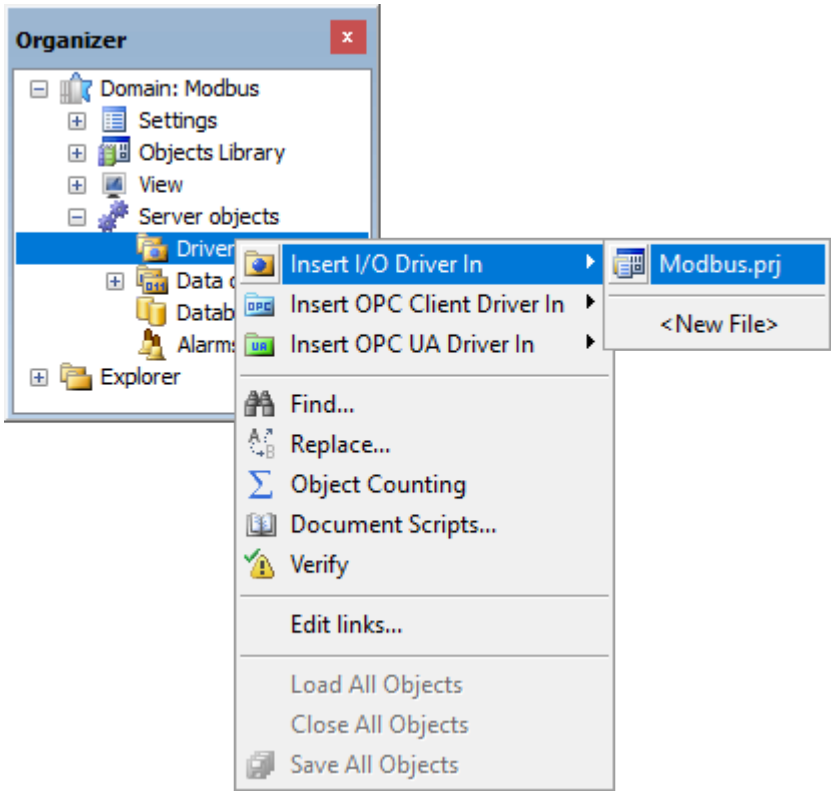
Depois, consulte o **passo seguinte** deste Manual, que ensina como configurar este Driver na janela de configuração para os casos mais comuns.

Adicionando este Driver em uma Aplicação da Elipse Software

Esta seção descreve como adicionar este Driver em aplicações **Elipse E3, Elipse Power ou Elipse Water** e **Elipse SCADA**.

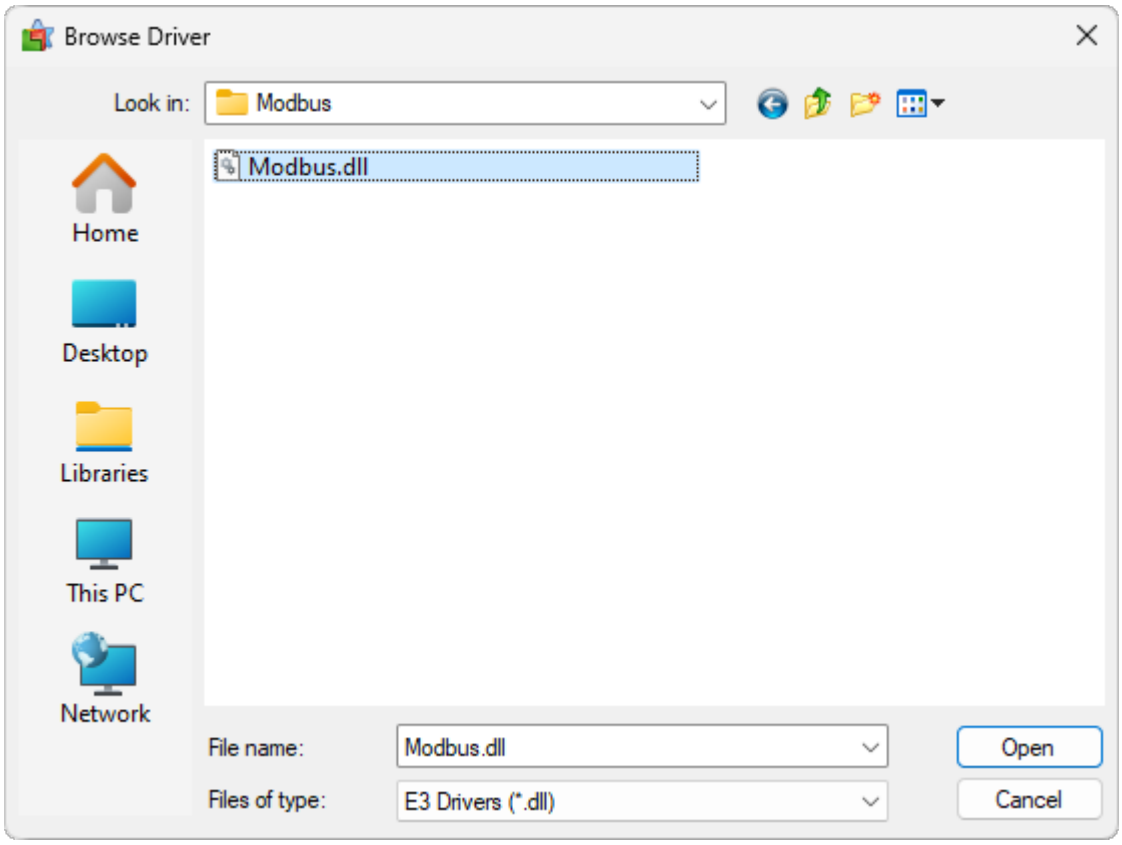
Elipse E3, Elipse Power ou Elipse Water

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.



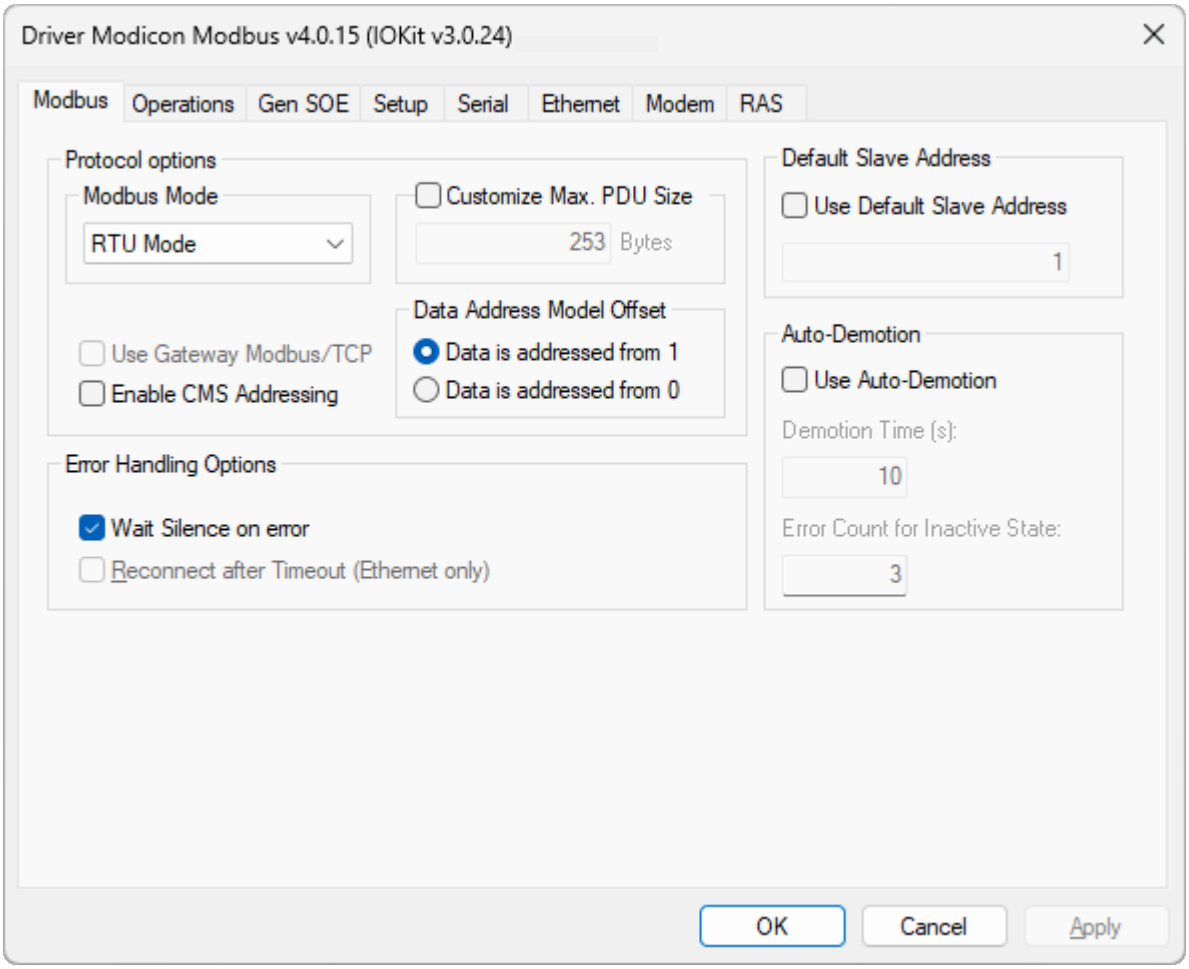
Adicionar um Driver a uma aplicação Elipse E3, Elipse Power ou Elipse Water

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



Selecionando um Driver

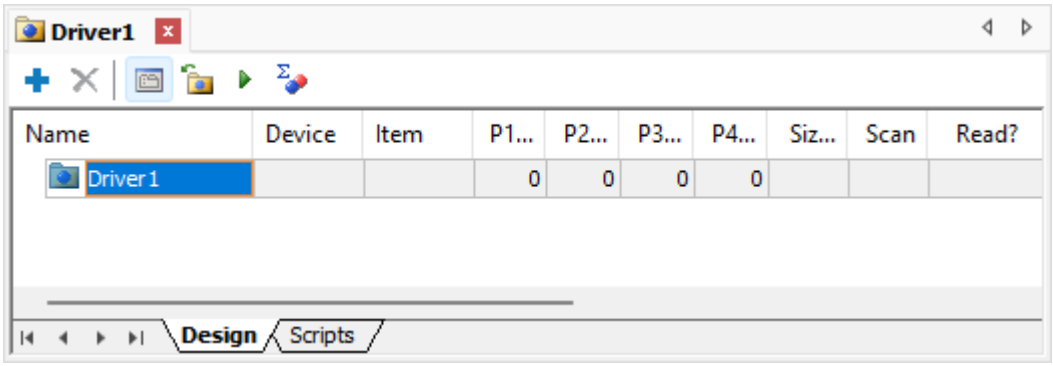
A janela de configurações deste Driver abre-se automaticamente, conforme mostrado na figura a seguir.



Janela de configurações deste Driver

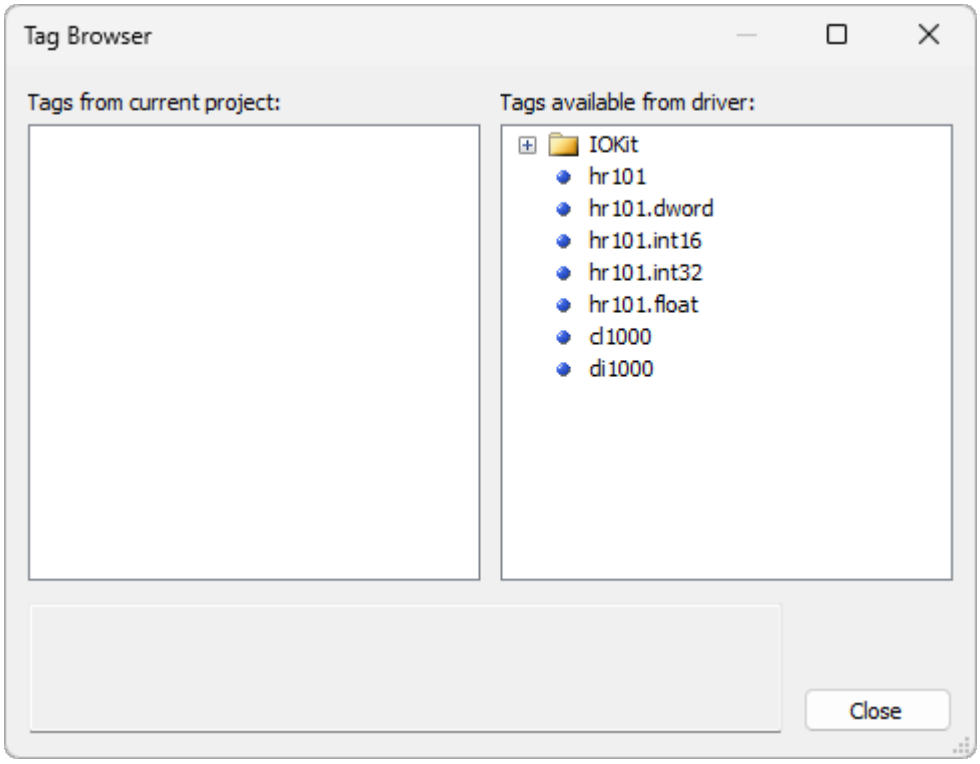
No **segundo passo** do tópico **Guia de Configuração Rápida** é apresentado o procedimento para a configuração básica deste Driver, para os casos de uso mais comuns. No tópico **Propriedades** a configuração é apresentada em detalhes.

A janela de configuração deste 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 deste Driver, clique em **OK** para que se abra a janela Tag Browser, permitindo inserir em uma aplicação Tags pré-definidos, com base nas configurações mais utilizadas. A figura a seguir mostra a janela 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*).

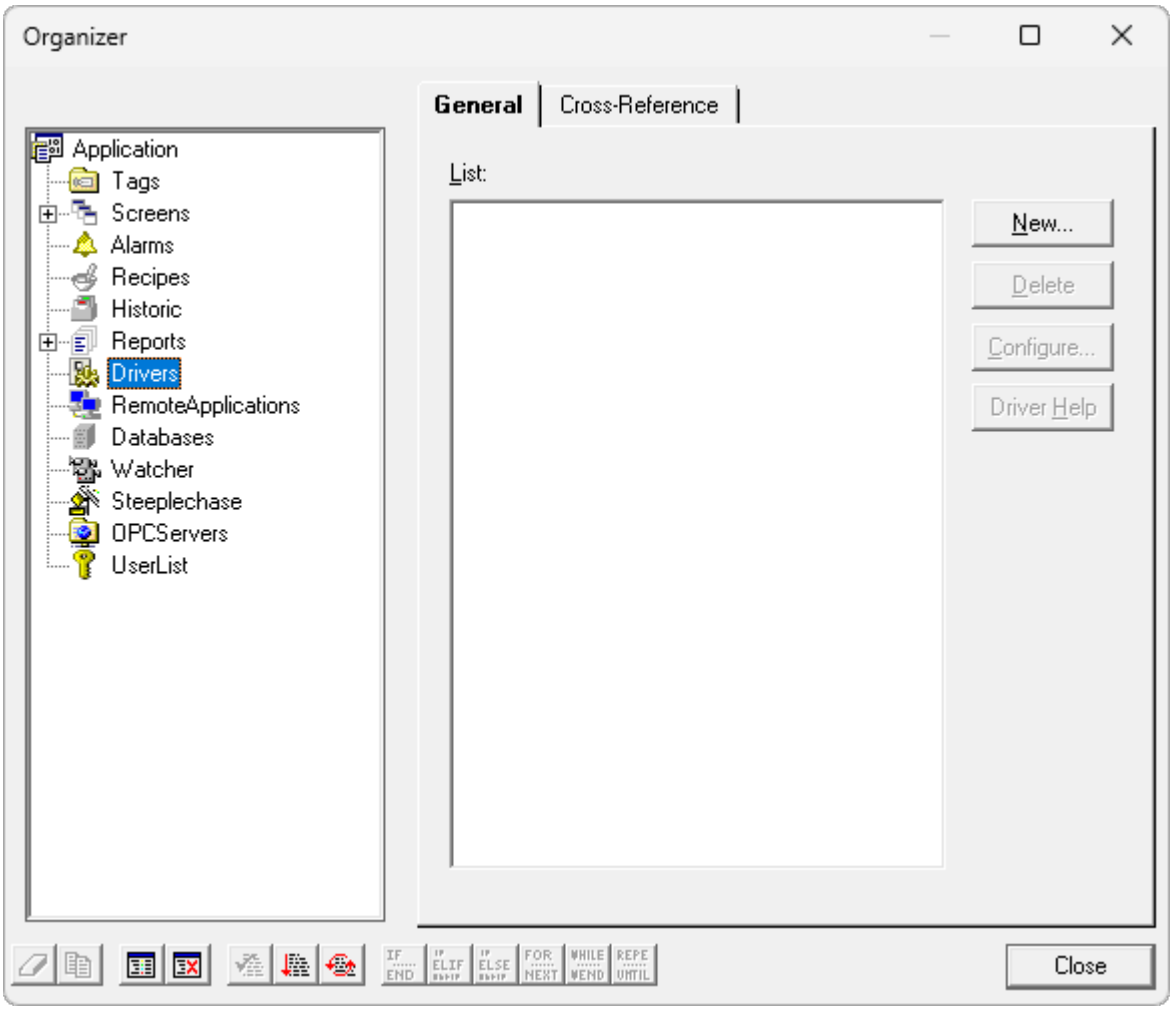


Janela Tag Browser

Os Tags disponíveis na janela Tag Browser são Tags **configurados usando Strings**, novo método que não utiliza o antigo conceito de operações. Insira os mais adequados em uma aplicação, editando os campos conforme a necessidade. A janela Tag Browser pode ser aberta posteriormente clicando-se em **Tag Browser** .

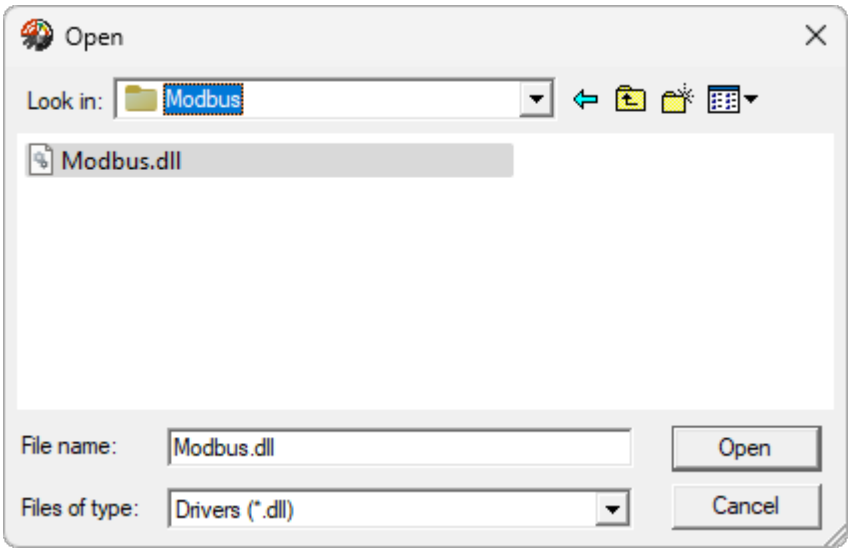
Elipse SCADA

No Organizer, selecione o item **Drivers** e clique em **New** (*Novo*).



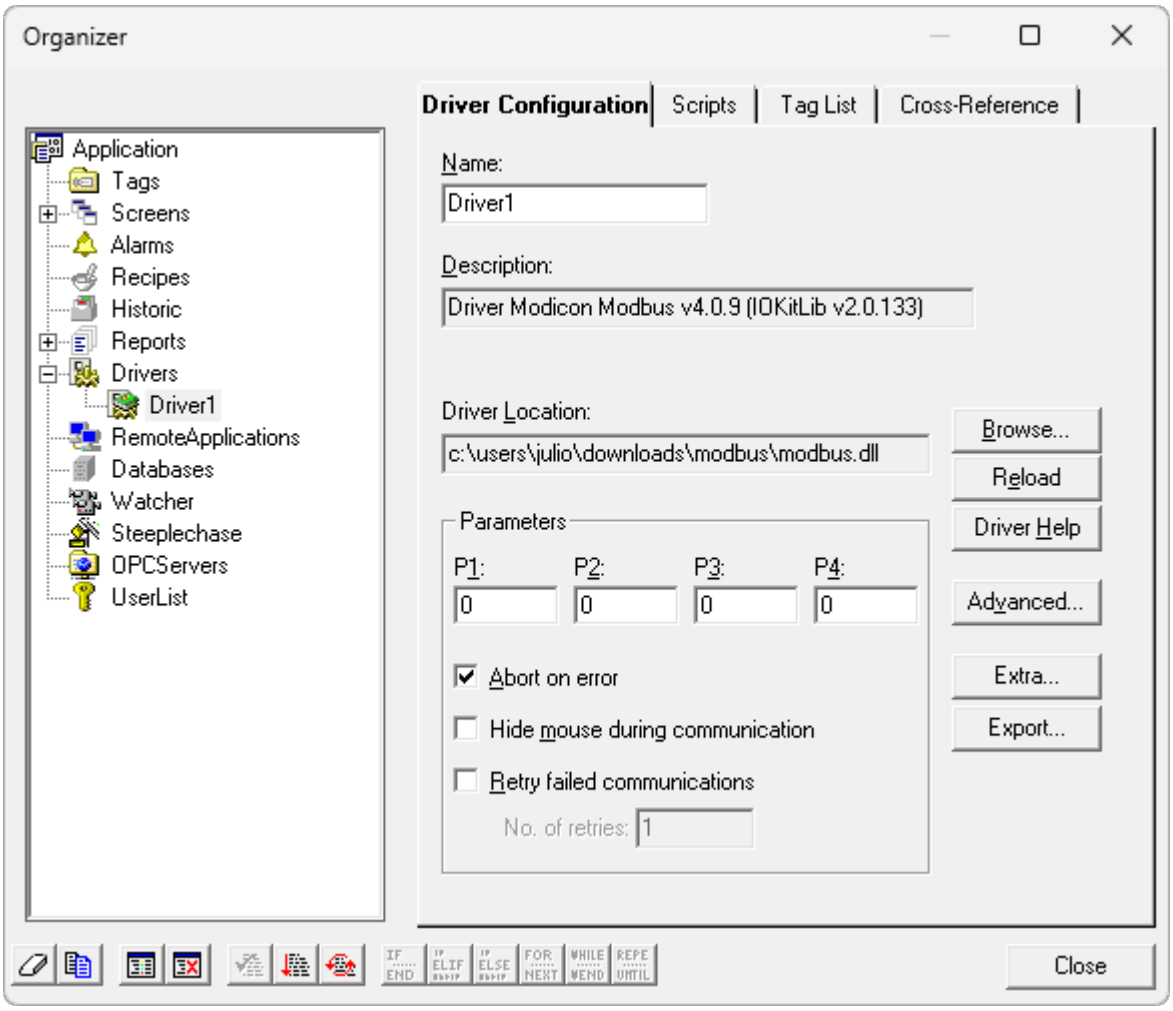
Adicionar um novo Driver ao Elipse SCADA

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



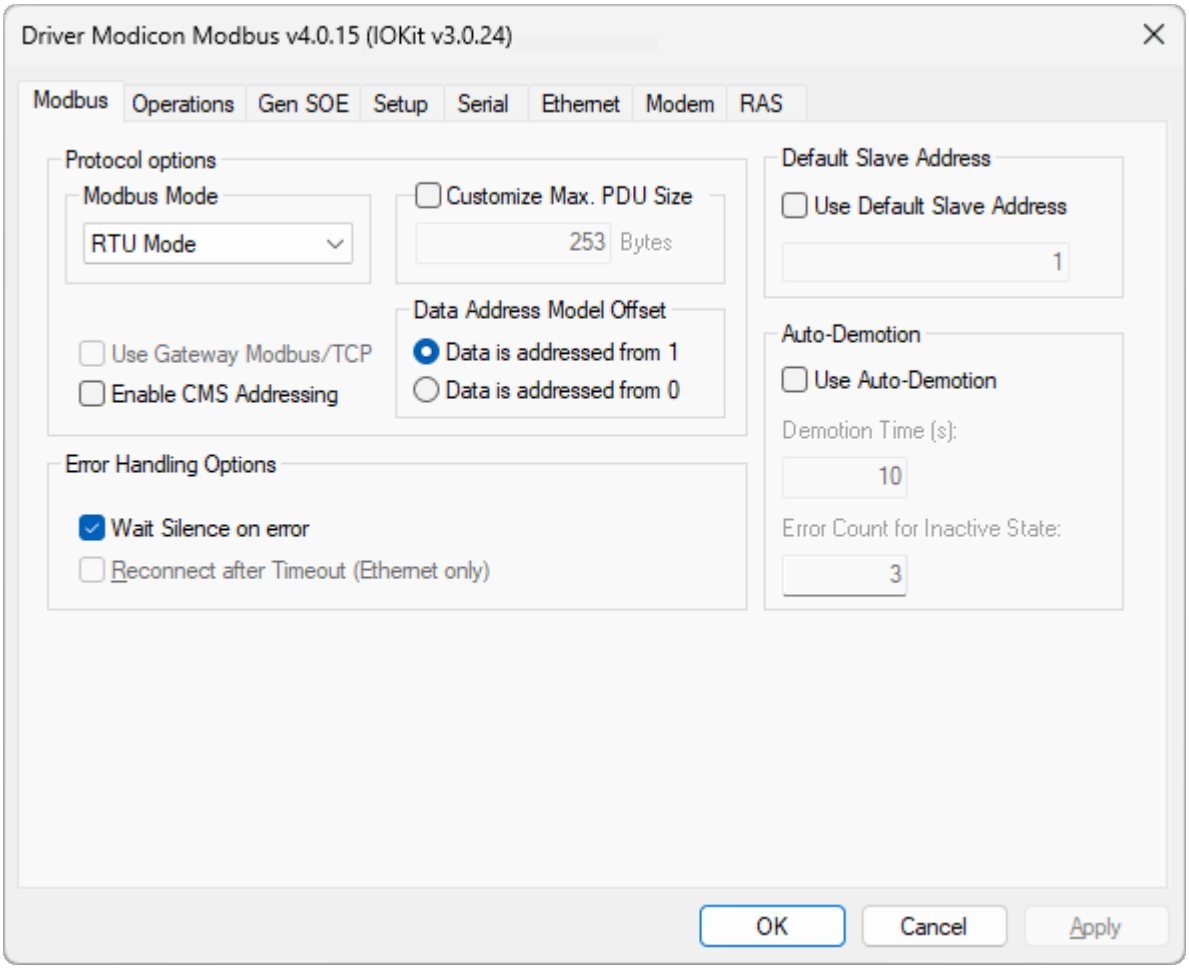
Selecionando um Driver

Este Driver é então adicionado em uma aplicação.



Driver no Organizer

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



Janela de configuração deste Driver

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

Configurando este Driver

Após inserir este Driver em uma aplicação, deve-se abrir a janela de configurações, conforme já explicado nos tópicos **Elipse E3, Elipse Power ou Elipse Water** ou **Elipse SCADA**. Na janela de configuração, siga estes passos:

1. Configure a camada física de comunicação:

   a. Na aba **Setup**, selecione a camada física, ou seja, **Serial**, **Ethernet**, **Modem** ou **RAS**, a ser utilizada na conexão com um equipamento.

   b. Configure a camada física selecionada na aba correspondente, ou seja, **Serial**, **Ethernet**, **Modem** ou **RAS**.

   c. Se precisar de mais informações sobre a configuração da camada física, consulte o tópico **Documentação das Interfaces de Comunicação**.

2. Na aba **Modbus**, selecione o modo do protocolo (**RTU**, **ASCII** ou **TCP**) utilizado por um 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 as configurações padrão. Caso precise de mais informações sobre as opções desta aba, consulte o tópico **Aba Modbus**.

NOTA

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

3. Caso se esteja criando uma aplicação em produtos mais recentes da **Elipse Software**, tais como o **Elipse E3**, o **Elipse Power** ou o **Elipse Water**, é possível utilizar a configuração de Tags por **Strings** (campos **Dispositivo** e **Item**). Neste caso, vá para o **passo seguinte** deste guia.

4. Caso precise ainda utilizar a antiga configuração numérica, por parâmetros *N* ou *B*, usada no **Elipse SCADA**, é importante examinar a aba **Operations**. Observe as operações padrão já pré-configuradas neste Driver. Estas operações são configurações de funções e formatações de dados que são posteriormente referenciadas pelos Tags de uma aplicação. Estas operações padrão, já disponíveis quando este 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 são necessárias para uma aplicação. Caso as operações pré-definidas não se enquadrem nas necessidades, é necessário editá-las ou mesmo criar novas operações. Se este é o caso, consulte o tópico **Aba Operations**. A tabela a seguir lista as operações pré-definidas.

Operações pré-definidas

OPERAÇÃO	LEITURA	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
5	3: Read Holding Registers	16: Write Multiple Registers	**Float**	Leitura de valores com ponto flutuante de 32 bits
6	1: Read Multiple Coils	15: Write Multiple Coils	**Bit**	Leitura e escrita de bits
7	2: Read Discrete Inputs	None	**Bit**	Leitura de bits de um bloco de dados de Entradas Discretas (*Discrete Inputs*)

NOTA

As operações padrão estão configuradas partindo do princípio que um 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 um 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 deste Driver, consulte 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 **Elipse E3**, no **Elipse Power** ou no **Elipse Water** e no antigo **Elipse SCADA** para os usos mais comuns.

Configuração de Tags no Elipse E3, no Elipse Power ou no Elipse Water

A configuração de Tags no **Elipse E3**, no **Elipse Power** ou no **Elipse Water** 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 de uma aplicação e facilita a manutenção.

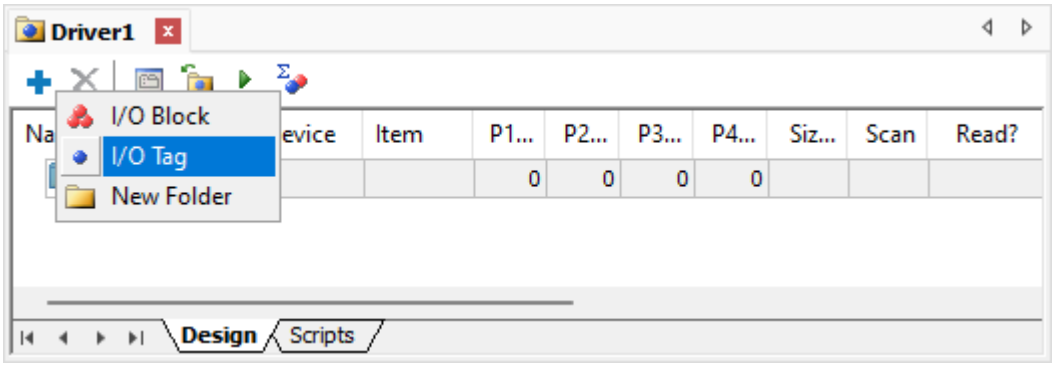
A seguir estão descritos os procedimentos recomendados para a configuração por **Strings** e também para a antiga configuração numérica, caso seja necessária para aplicações legadas.

Configuração por Strings

Para adicionar Tags configurados por **Strings**, o usuário tem a opção de importar modelos pré-definidos da janela Tag Browser, conforme explicado no tópico **Adicionando este 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 a janela Tag Browser clicando em **Tag Browser** .

Para adicionar um Tag em uma aplicação sem a janela Tag Browser, siga estes passos:

1. No Organizer, clique duas vezes neste 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 este Tag no campo **Name** (*Nome*). Clique em **OK** para criar este Tag.

3. Na coluna **Device** (*Dispositivo*), digite o valor numérico do *Slave Id* de um equipamento para comunicação, 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, e é utilizado apenas o endereço IP e a porta TCP/IP configurada na aba **Ethernet**, que deve constar na documentação deste 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", em que 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).

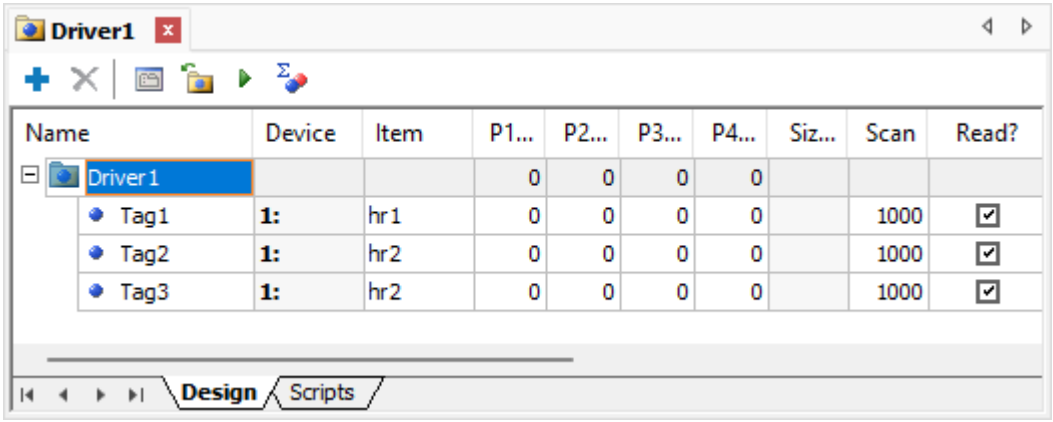
d. Leitura ou escrita do *Coil* de endereço FFF0h (65520) usando as funções **01** e **05** (escrita de bits simples, um a um): **Item** deve ser igual a "scl65520" ou "scl&hFFF0" (o prefixo "&h" pode ser usado para fornecer endereços em formato hexadecimal).

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 de um equipamento, que deve constar na documentação do fabricante. Em caso de dúvida, consulte o tópico **Dicas de Endereçamento (Modbus Convention)**.

Configure preferencialmente Tags simples, chamados de Tags PLC no antigo **Elipse SCADA**, ao invés de Tags Bloco, mantendo o recurso de Superblocos habilitado, ou seja, a propriedade **EnableReadGrouping** configurada como Verdadeiro, deixando a cargo de uma aplicação e a este Driver a otimização deste agrupamento. Para mais detalhes, consulte o tópico **Leitura por Superblocos (Agrupamento)**.

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

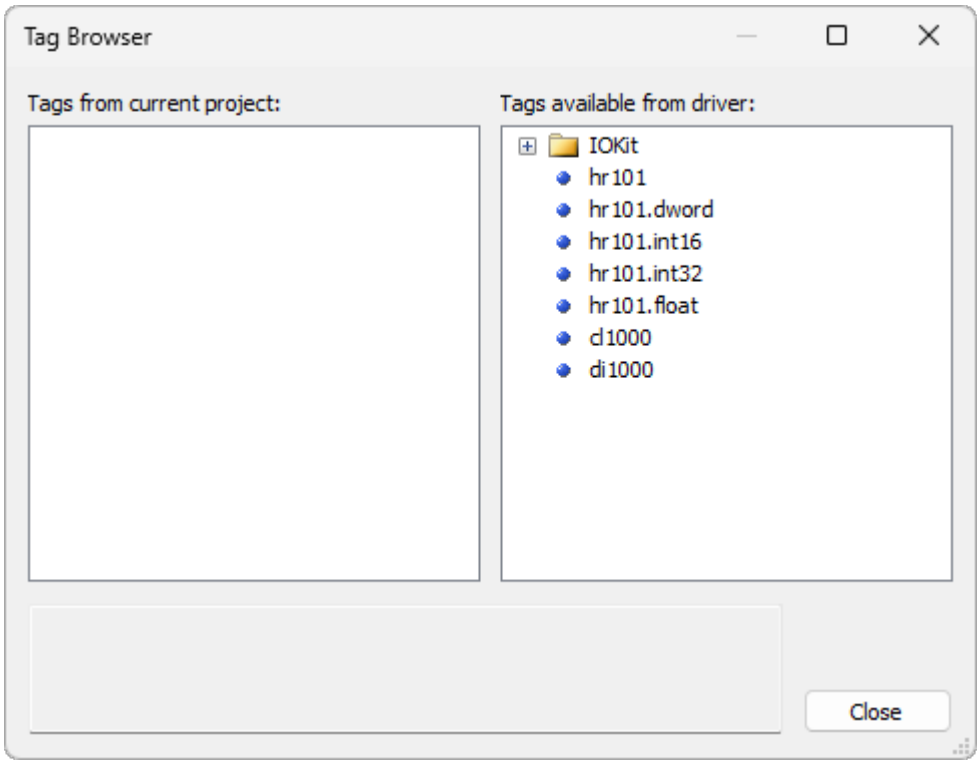


Exemplo de Tags configurados por Strings

Configuração Numérica

No caso do **Elipse E3**, do **Elipse Power** ou do **Elipse Water**, é possível usar a janela 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 Tag Browser, mostrada na figura a seguir, é aberta ao clicar-se em **OK** ao fechar a janela de configurações deste Driver.



Tag Browser para a configuração numérica de Tags

Para adicionar um Tag a uma 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 **Elipse E3, Elipse Power ou Elipse Water**. Para muitos equipamentos, a operação **1** (um), 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 uma 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 a janela Tag Browser e configure o parâmetro *N4* ou *B4* de cada Tag com o endereço de registro ou bit a ser lido ou escrito, conforme o mapa de registradores de um equipamento. Este mapa de endereços deve constar na documentação do fabricante. Em caso de dúvida, consulte o tópico **Dicas de Endereçamento (Modbus Convention)**.

3. Configure também o parâmetro *N1* ou *B1* de cada Tag com o endereço de um equipamento (*Slave Id*) a ser acessado em cada caso. Este parâmetro em geral é configurável em um 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 uma aplicação.

Configure preferencialmente Tags simples, chamados de Tags PLC no antigo **Elipse SCADA**, ao invés de Tags Bloco, mantendo o recurso de Superblocos habilitado, ou seja, a propriedade **EnableReadGrouping** configurada como Verdadeiro, deixando a cargo de uma aplicação e a este Driver a otimização deste agrupamento. Para mais detalhes, consulte o tópico **Leitura por Superblocos (Agrupamento)**.

Configuração de Tags no Elipse SCADA

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

- **N1 ou B1**: Endereço de um equipamento (*Slave Id*)
- **N2 ou B2**: Código de uma operação

- **N3 ou B3**: Não usado, deixe em 0 (zero)
- **N4 ou B4**: Endereço de um 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* ou *B4*, consulte o tópico **Dicas de Endereçamento (Modbus Convention)**.

Uma vez que o **Elipse 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 deste protocolo.

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

Considerações Finais

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

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

Mais 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, e hoje é um padrão aberto, mantido pela Organização Modbus (modbus.org), e implementado por centenas de fabricantes em milhares de equipamentos. A Schneider Electric, atual controladora da Modicon, transferiu os direitos deste protocolo para a Organização Modbus em abril de 2004, firmando o compromisso de manter este protocolo aberto. A especificação pode ser obtida gratuitamente no site da Organização, *www.modbus.org*, e a utilização deste protocolo é livre de taxas de licenciamento.

Este protocolo é baseado em mensagens de comando e resposta, posicionado no nível 7 (sete) do modelo OSI (*Open Systems Interconnection*), a 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 8 (oito) bits. As categorias de códigos de funções existentes são as seguintes:

- **Códigos de funções públicas**: Funções bem definidas por este 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, e que podem 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 deste protocolo, versão **1.1b**, que está disponível no *site oficial deste protocolo*

Este Driver implementa 11 das 19 funções públicas previstas na versão atual, **1.1b**, da especificação deste protocolo, bem como algumas funções específicas de fabricantes ou relacionadas a recursos específicos deste Driver, denominadas **Funções Especiais**. As funções públicas implementadas estão descritas no tópico **Funções Suportadas**.

Este Driver suporta comunicação segura em conformidade com a especificação **Modbus/TCP Security**, mantida pela Organização Modbus, que adiciona uma camada de segurança **TLS 1.2** sobre o transporte Modbus TCP tradicional, incluindo validação real da cadeia de certificados, verificação da identidade de um equipamento remoto e restrição às *cipher suites* exigidas por esta especificação. Para habilitar, selecione a opção **Use SSL** na aba **Ethernet** da janela de propriedades deste Driver e, na opção **SSL Settings**, informe o certificado da **Autoridade Certificadora (CA)** utilizado para validar a identidade de um equipamento remoto. Quando um equipamento remoto exige autenticação mútua (**mTLS**), informe também um certificado e chave privada próprios nos campos **Certificate Path** e **Private Key**.

Sites Recomendados

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

Mais informações referentes ao protocolo Modbus podem ser obtidas em *www.modbus.org,* site oficial deste protocolo.

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

O Simulador Modbus Slave **Modsim**, provavelmente o mais conhecido da categoria, pode ser adquirido em *www.win-tech.com/html/modsim32.htm*. Este software emula um equipamento, permitindo a comunicação com este Driver.

Existe também a alternativa gratuita **Free Modbus PLC Simulator**, disponível para download no site *www.plcsimulator.org*.

Outras alternativas de simuladores e outras ferramentas relacionadas a este protocolo podem ser encontradas no *site oficial deste 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 deste protocolo, como já referido, este Driver também implementa funções especiais, não definidas por este protocolo, em geral relacionadas à leitura da memória de massa. A lista das funções especiais suportadas pode ser consultada no tópico **Funções Especiais**. A configuração completa deste Driver está descrita no tópico **Configuração**.

Caso identifique a necessidade de adicionar suporte a alguma função 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 não definidas pelo protocolo Modbus padrão. Estas funções foram desenvolvidas para atender particularidades exclusivas de determinados equipamentos, ou também para disponibilizar, de forma padronizada por este Driver, recursos não disponíveis no protocolo padrão. Este Driver, na atual versão, inclui as funções especiais descritas a seguir.

Funções de Leitura

- **65 03**: Leitura da Memória de Massa (*ABB MGE 144*), descrita em mais detalhes no tópico **Leitura de Registros da Memória de Massa de Medidores ABB MGE 144**
- **GE SOE**: Leitura de eventos (*GE PAC RX7 Systems*), descrita em mais detalhes no tópico **Leitura de Buffer de Eventos em Controladores GE PAC RX7**
- **SP SOE**: Leitura de eventos (*Relés Schneider Electric da série SEPAM*), descrita em mais 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 em um equipamento **Escravo** (CLP), descrita em mais detalhes no tópico **Algoritmo de Leitura de SOE Genérico da Elipse Software**

Funções de Escrita

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

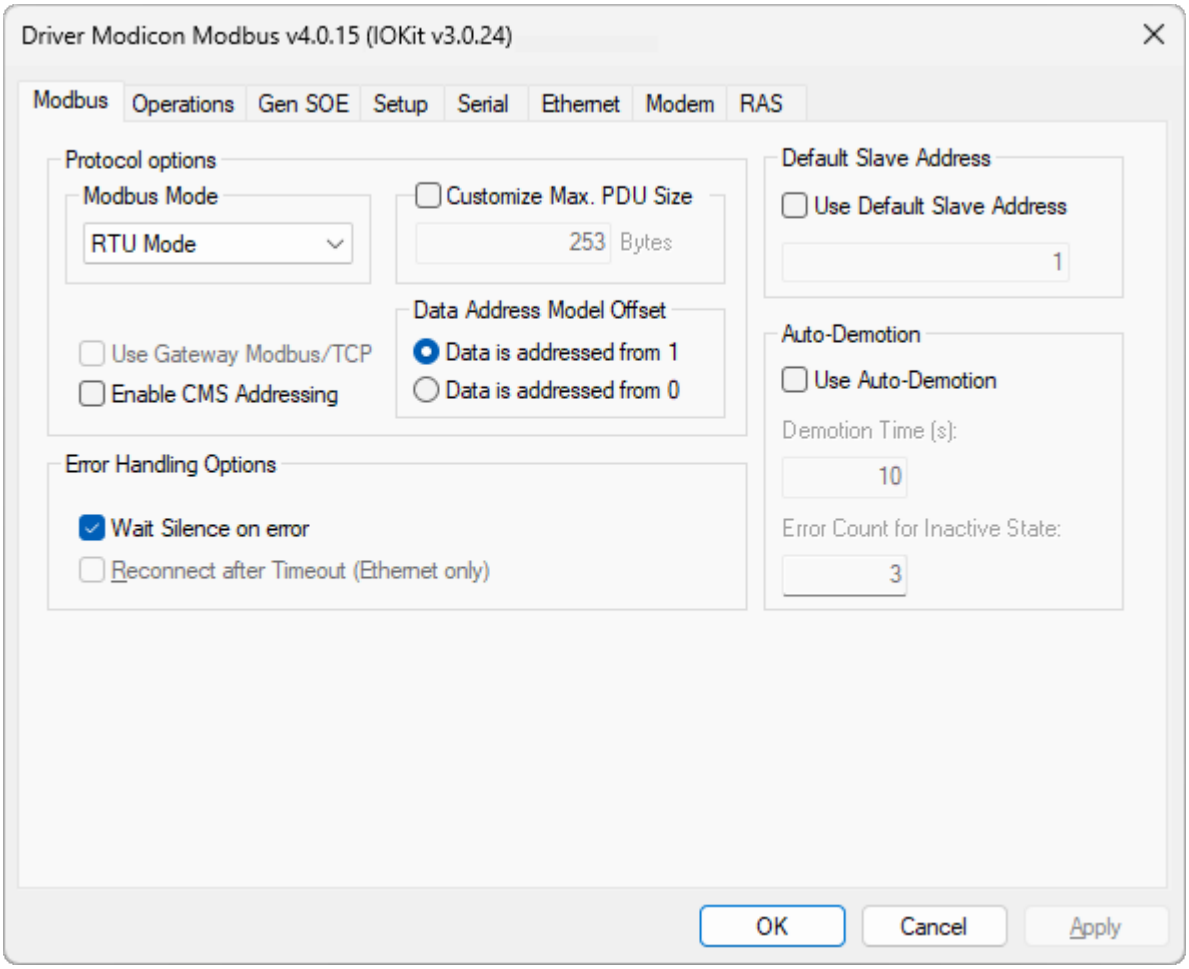
Configuração

Esta seção descreve como configurar este Driver. São abordados os tópicos **Propriedades**, **Configurando Tags** e **Leitura de Memória de Massa**.

Propriedades

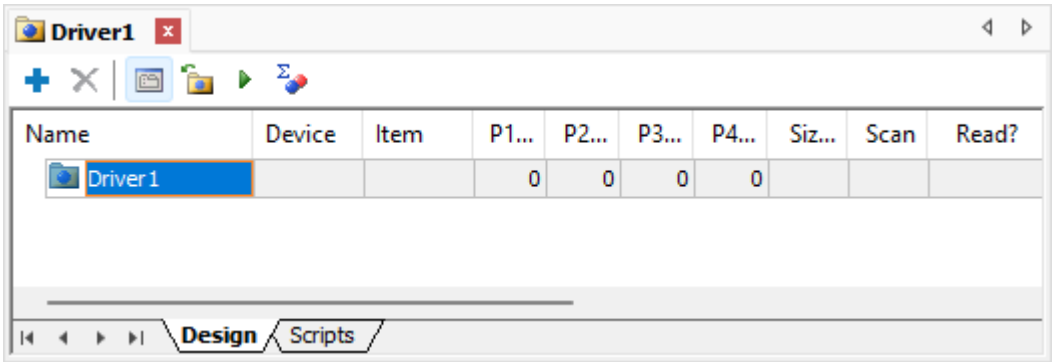
As propriedades deste Driver podem ser configuradas em tempo de configuração (*design time*) ou em tempo de execução (*run time*). A configuração em tempo de execução (*run time*) também é chamada de **Configuração em Modo Offline** e é descrita em um tópico específico.

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



Janela de configuração deste Driver

Para abrir a janela de configuração deste Driver no **Elipse E3**, no **Elipse Power** ou no **Elipse Water**, 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

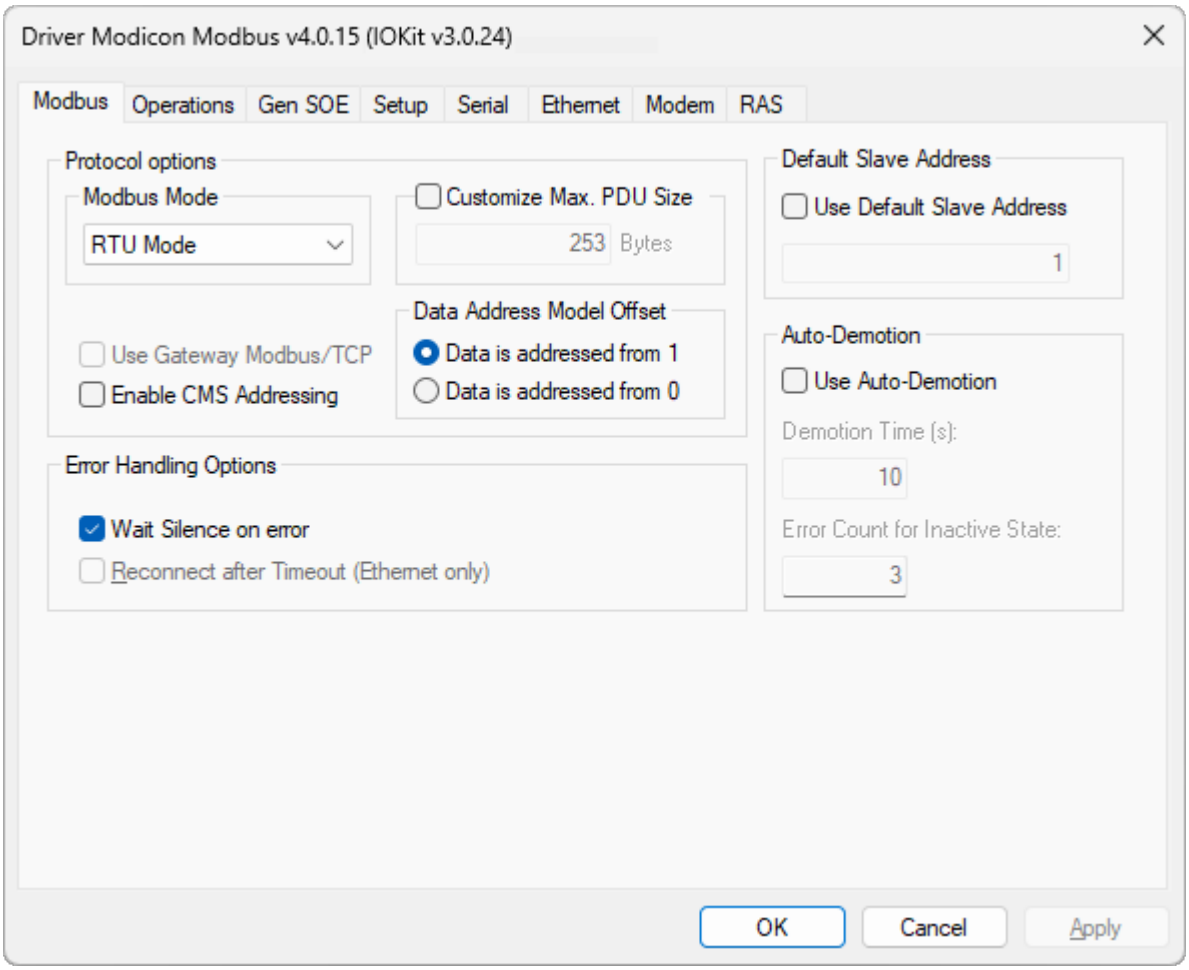
Já no **Elipse SCADA**, a janela de configuração deste Driver pode ser aberta clicando-se em **Extras**, no Organizer de uma aplicação.

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

Os tópicos **Aba Modbus**, **Aba Operations**, **Aba Gen SOE** e **Configuração em Modo Offline** descrevem as abas específicas deste Driver e também a configuração em tempo de execução, o chamado **Modo Offline**, utilizando scripts.

Aba Modbus

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



Aba Modbus da janela de configurações deste 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 Modbus, conforme a tabela a seguir.

Opções de protocolo disponíveis na aba Modbus

OPÇÃO	DESCRIÇÃO
Modbus Mode	Nessa opção 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, tais como **Serial**, **Ethernet TCP/IP**, ou **RAS**,

OPÇÃO	DESCRIÇÃO
	entre outros. As opções disponíveis são **Modo RTU**: Modo padrão para uso em comunicação serial e que inclui CRC de 16 bits, **Modo ASCII**: Também usado em comunicação serial, é utilizado em equipamentos mais simples, que não suportam os requisitos do modo **RTU**. Utiliza caracteres ASCII para transmissão, onde cada byte contém 2 (dois) caracteres ASCII, um por *nibble*, e por isto menos eficiente que o modo **RTU** e mais raramente encontrado no mercado. Usa LRC (*Longitudinal Redundancy Checking*) para a verificação de erros ou **Modo ModbusTCP**: Usado para comunicação em modo **TCP/IP**. 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 este Driver assuma como resposta válida para o comando atual os *frames* de resposta a comandos anteriores. Esta situação pode ocorrer se os modos anteriores são encapsulados em TCP/IP
Customize Max. PDU Size	Se habilitada, esta opção permite definir um tamanho máximo personalizado para o PDU (*Protocol Data Unit*). O PDU é a parte do protocolo que não varia entre os modos **ModbusTCP**, **ASCII** e **RTU**, 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 **valor padrão definido pelo protocolo Modbus versão 1.1b**, de 253 bytes. Esta é a opção recomendada para a maioria dos equipamentos
Use Gateway Modbus/TCP	Em modo **Ethernet**, via **Modbus TCP**, esta opção indica a presença de um *Gateway Modbus* para acesso a uma rede Serial RS-485, reservando o *Slave ID* com valor 0 (zero) para o modo **Broadcast**. Caso seja uma comunicação direta com um equipamento em modo **Ethernet**, desabilite esta opção se utilizar um *Slave ID* com valor 0 (zero)
Enable CMS Addressing	Esta opção deve ser usada apenas em equipamentos que suportem o protocolo TeleBUS. Se habilitada, este Driver passa a aceitar um **Word** de 16 bits como endereço de um **Escravo**, ou seja, passa a ser possível definir valores acima de 255 e abaixo de 65535 como endereço de um **Escravo**. Neste caso, o endereço de um **Escravo** passa a ser definido no protocolo por 3 (três) bytes. Além disto, a opção **Default Slave Address** passa a não funcionar mais
Data Address Model Offset	Esta opção habilita ou desabilita o *offset* de dados padrão do protocolo, de 1 (uma) unidade. As opções disponíveis são **Data is addressed from 1 (padrão)**: O endereço fornecido (endereço do campo **Item** na configuração por **Strings** ou o parâmetro *N4* ou *B4* na configuração numérica) é decrementado em 1 (um) antes de ser enviado a um equipamento. Este *offset* é previsto na especificação do protocolo, e portanto esta é a opção padrão ou **Data is addressed from 0**: O endereço fornecido pelo usuário é usado nas requisições do protocolo, sem alterações. Como regra geral, selecione a primeira opção caso o

OPÇÃO	DESCRIÇÃO
	mapa de registradores de um equipamento inicie em 1 (um) e a segunda caso inicie em 0 (zero). Verifique também se o fabricante usa *offsets* adicionais da antiga **Modbus Convention**. Para mais informações, consulte a **seção a seguir**

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 é inevitável a utilização de **Modbus RTU** em meio **Ethernet TCP/IP**, não deixe de habilitar a opção **Reconnect after Timeout**, descrita 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.

O modelo de dados padrão deste protocolo define os 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 em um PDU e o endereço dos elementos de dados, portanto, possui um deslocamento (*offset*) de 1 (um), ou seja, se em um 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 selecionar se deseja que este Driver ajuste o valor automaticamente, de forma a permitir o uso do endereço do elemento de dado nos Tags, a opção padrão, ou se deseja que o valor enviado em um PDU seja o mesmo valor fornecido na configuração dos Tags, o parâmetro *N4* ou *B4* na **configuração numérica**. Existem equipamentos que seguem o padrão Modbus nos mapas de endereços, iniciando em 1 (um), e outros que mapeiam os dados sem o *offset* padrão, usando diretamente o valor de endereço presente no *frame* de comunicação, iniciando em 0 (zero).

Além deste *offset* unitário, existem ainda equipamentos que utilizam o antigo padrão de *offsets* utilizado pela Modicon, empresa criadora deste protocolo, padrão conhecido como **Modbus Convention**, detalhado no tópico **Dicas de Endereçamento (Modbus Convention)**. Consulte no manual de um 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 **2.3** deste Driver, em que 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 deste Driver.

Outras opções disponíveis na aba Modbus

OPÇÃO	DESCRIÇÃO
Default Slave Address	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 *Slave Id*, o parâmetro *N1* ou *B1* na **configuração numérica** ou o campo **Dispositivo** na **configuração por Strings**, em 1000, ou seja, todos os Tags com *Slave Id* igual a 1000 têm este valor substituído pelo valor configurado na opção **Default Slave Address**. Também é possível forçar o uso do endereço padrão em todos os Tags, independente do valor de *Slave Id* configurado, selecionando-se a opção **Use Default Address**
Wait Silence on error	Se esta opção está habilitada, após cada erro de comunicação este Driver permanece em *loop*, recebendo dados até que ocorra um *time-out*. 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 deste erro, e que possam ser confundidos com uma resposta a um novo comando
Reconnect after Timeout (Ethernet only)	Com esta opção habilitada, após qualquer erro de *time-out* na recepção de *frames* de um equipamento, este Driver promove a desconexão e a reconexão da camada física, limpando a conexão de possíveis *frames* atrasados que estejam em tráfego e que possam afetar futuras requisições. Esta opção deve sempre ser habilitada caso seja inevitável o uso de **Modbus RTU** em meio **Ethernet TCP/IP** em sistemas legados, uma vez que o modo **RTU** não possui controle de transação, portanto nem sempre é possível distinguir um *frame* de resposta correto de um outro *frame* atrasado resultante de um leitura anterior, possivelmente de outro endereço, e que tenha falhado por *time-out*. Para novas aplicações, recomenda-se fortemente que **NÃO** sejam utilizados os modos **Modbus RTU** ou **Modbus ASC** em meio **Ethernet TCP/IP**. Note que é preciso manter habilitada a opção **Retry failed connection every** da aba **Setup** da biblioteca **IOKit** para que este Driver se reconecte após este *time-out*. Caso contrário, este *time-out* apenas gera uma desconexão e cabe à aplicação o gerenciamento desta nova conexão
Use Auto-Demotion	Com esta opção habilitada, este Driver comunica com todos os equipamentos **Escravos** pelo modo **Auto-Demotion**
Demotion Time	Define o tempo, em segundos, que um equipamento **Escravo** permanece inativo na comunicação após exceder o limite de erros por *time-out*, definido na opção **Error Count for Inactive State**. Após expirar este tempo, um equipamento **Escravo** realiza uma nova tentativa de comunicação. Se esta tentativa falha novamente por *time-out*, aplica-se um novo tempo de inatividade, mantendo-se em modo **Auto-Demotion**. Se um equipamento **Escravo** volta a comunicar ou falha por outros motivos, este

OPÇÃO	DESCRIÇÃO
	equipamento volta ao estado normal de comunicação, retirando-se do modo **Auto-Demotion**
Error Count for Inactive State	Define quantas vezes consecutivas um equipamento **Escravo** deve falhar por *time-out*, antes de permanecer temporariamente inativo pelo tempo definido na opção **Demotion Time**. Quando este equipamento **Escravo** volta a comunicar ou falha por outros motivos, este contador de falhas por *time-out* é zerado

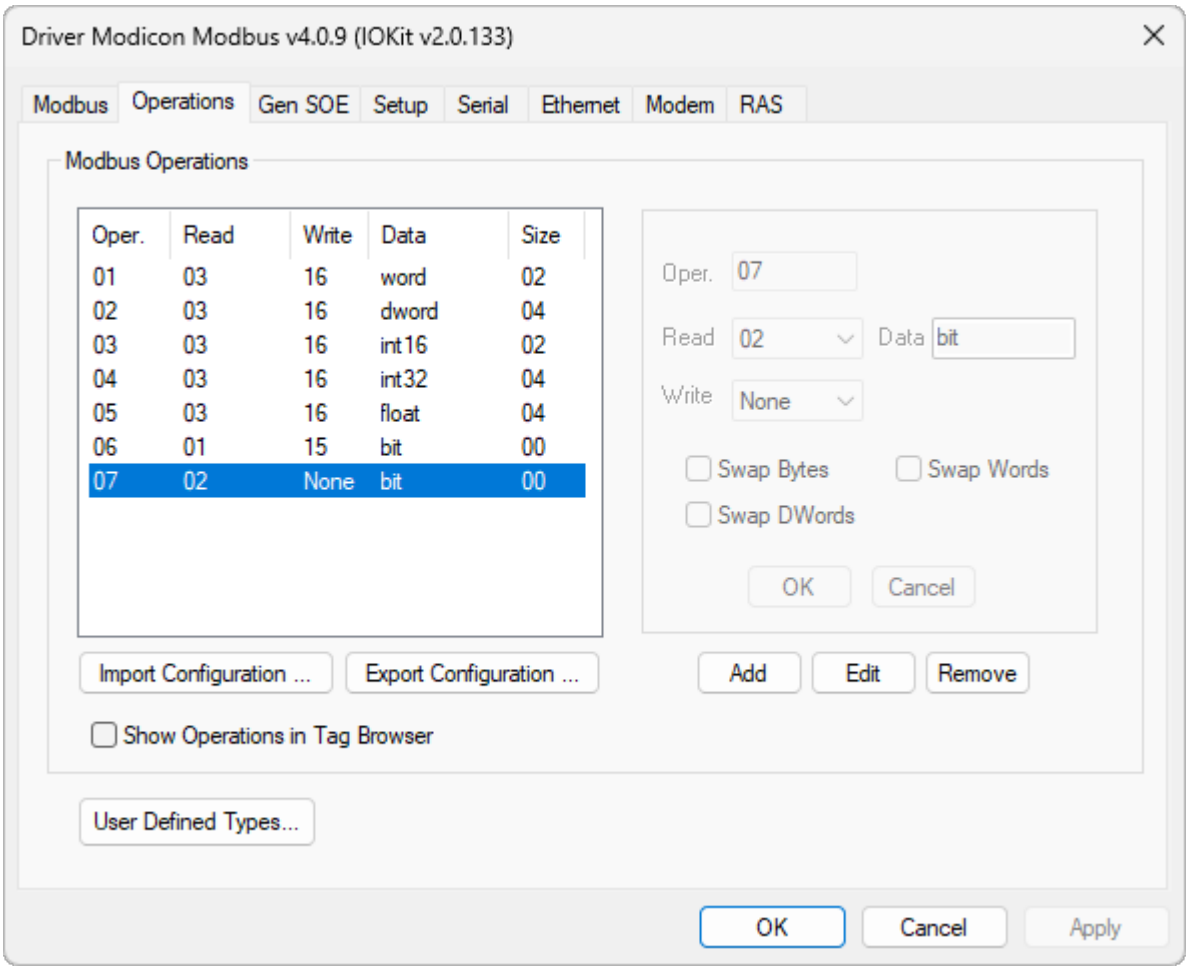
NOTA

A antiga opção **Swap Address Delay** foi removida da janela de configuração na versão **2.8**. Este Driver ainda mantém suporte a esta opção em aplicações pré-existentes e permite habilitá-la por scripts. Para mais informações, consulte o tópico **Configuração em Modo Offline**. Para aplicações recentes, recomenda-se utilizar a opção **Inter-frame Delay** da aba **Serial** da biblioteca **IOKit**, que substitui esta antiga opção com vantagens.

Aba Operations

Este tópico descreve a configuração da aba **Operations** da janela de configuração deste 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**, pois é usada apenas na antiga **configuração numérica**, os parâmetros *N* ou *B*, do **Elipse SCADA**.



Aba Operations na janela de configurações deste 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 em um 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 de uma aplicação. Em outras palavras, neste Driver 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 destas 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**. É possível, e em geral necessário, atribuir uma mesma operação a diversos Tags, que têm em comum o mesmo valor nos parâmetros *N2* ou *B2*.

Funções

O protocolo Modbus define funções de leitura e escrita, as quais podem acessar espaços de endereçamento distintos em um 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 por este Driver, e que podem ser atribuídas às operações configuradas, pode ser consultada no tópico **Funções Suportadas**.

Além das funções do protocolo, este 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 estão 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 de um equipamento, ou seja, para a leitura de um Tag associado a uma operação que defina o tipo de dados **Float** de 32 bits, este Driver precisa ler 2 (dois) registros consecutivos de 16 bits de um equipamento, concatená-los e realizar a conversão para o formato **Float**.

Também é possível definir tipos de dados de 8 (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 2 (dois) Elementos de Bloco de tipos de dados de 8 (oito) bits, este Driver é obrigado a acessar um registro distinto de 16 bits em um equipamento. Por este motivo, este Driver não permite a escrita de tipos de dados de 8 (oito) bits em Tags, em Elementos isolados de Bloco ou em Blocos de tamanho ímpar ou unitário. A escrita de tipos de dados de 8 (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 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 dados 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 os 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**. As situações possíveis são as seguintes:

- Caso um 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 um 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, todas as 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 um valor retornado por um 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 8 (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 um 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 um usuário deseja ler somente 1 (um) bit do valor retornado por um equipamento, mas não é possível usar o mapeamento de bits de uma aplicação.

Para a maioria dos usuários, os campos de mapeamento de bits de uma aplicação são a melhor alternativa para o acesso à máscaras de bits, e não é preciso recorrer a este recurso deste Driver.

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

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

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

Operações padrão

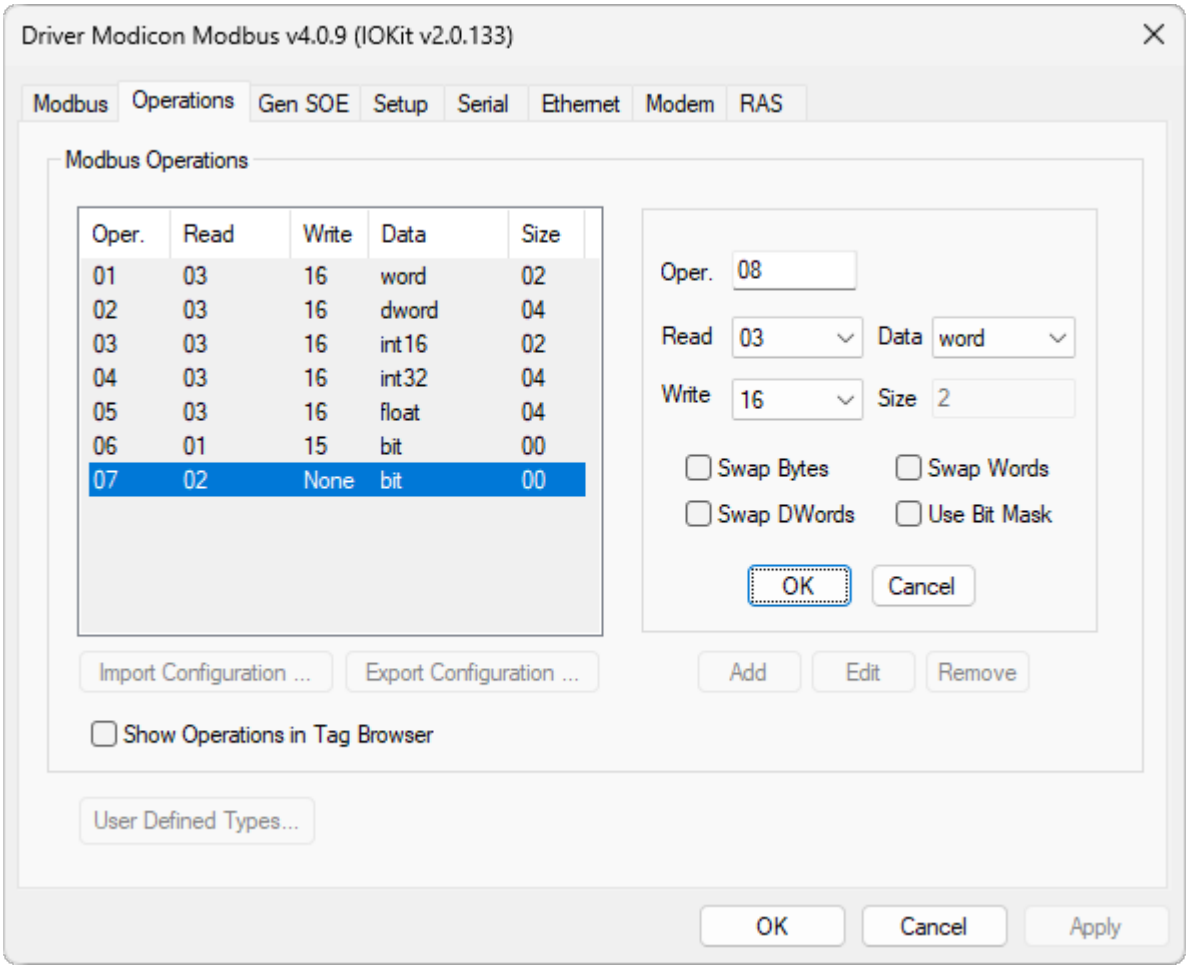
OPERAÇÃO	LEITURA	ESCRITA	TIPO DE DADOS	FINALIDADE
1	3 - Read Holding Registers	16 - Write Multiple Registers	**Word**	Leitura e escrita de inteiros de 16 bits sem sinal
2	3 - Read Holding Registers	16 - Write Multiple Registers	**DWord**	Leitura e escrita de inteiros de 32 bits sem sinal
3	3 - Read Holding 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	LEITURA	ESCRITA	TIPO DE DADOS	FINALIDADE
5	3 - Read Holding Registers	16 - Write Multiple Registers	**Float**	Leitura e escrita de valores de ponto flutuante de 32 bits
6	1 - Read Coil Status	15 - Write Multiple Coils	**Bit**	Leitura e escrita de bits
7	2 - Read Discrete Inputs	None	**Bit**	Leitura de bits do bloco de dados de Entradas Discretas (*Discrete Inputs*)

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

Definição de Operações

Para adicionar uma operação neste Driver, clique em **Add**.



Adicionando uma operação

Para configurar esta operação, selecione um número, utilizado no parâmetro *N2* ou *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 por este Driver. Note que, ao clicar em **Add**, este Driver já sugere um valor que ainda não esteja em uso para a operação.

Para mais informações sobre os tipos de dados suportados, consulte 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
Size	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 **Byte**, **Word** e **Int16**, devendo ser preenchido para tipos de dados **String** e **BCD**. No caso de **Strings**, este tamanho define exatamente o número de bytes enviados ou recebidos para cada valor **String**, isto é, para cada Tag ou Elemento de Bloco. Se a **String** 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 **String** neste Driver não possui um limite máximo de tamanho definido, este limite é o **máximo permitido pelo protocolo** para a área de dados do *frame* de uma determinada função
Swap Bytes	Indica que este Driver deve inverter a ordem dos bytes, um a um, para obter um valor
Swap Words	Indica que este Driver deve inverter a ordem dos bytes, dois a dois, em **Words**, para obter um valor
Swap DWords	Indica que este Driver deve inverter a ordem dos bytes, quatro a quatro, em **DWords**, para obter um valor
Use Bit Mask	Habilita o mascaramento de bits de registradores, através do parâmetro *N3* ou *B3*. 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, **Int16**, **Int32**, **Word** ou **DWord**. Operações com esta opção habilitada não podem ser usadas para escrita. Para a maioria dos usuários, recomenda-se usar o mapeamento de bits de uma aplicação, deixando esta opção desmarcada. Para mais informações, consulte a **seção específica**

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
Import Configuration	Esta opção permite importar configurações de operações de versões anteriores à versão **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 estas configurações, que agora são armazenadas no próprio arquivo de uma aplicação. Para mais detalhes, consulte o tópico **Importação e Exportação de Operações**

OPÇÃO	DESCRIÇÃO
Export Configuration	Esta opção executa a operação inversa da opção **Import Configuration**, 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 mais detalhes, consulte o tópico **Importação e Exportação de Operações**
Show Operations in Tag Browser	Se esta opção não está selecionada (padrão), são mostrados modelos de **Tags configurados por Strings** (campos **Dispositivo** e **Item**) na janela Tag Browser. Se está selecionada, modelos de **Tags configurados numericamente** (parâmetros *N* ou *B*), representando as operações configuradas, são mostrados na janela Tag Browser. Quando instâncias deste Driver são criadas, esta opção vem desmarcada por padrão. Em aplicações legadas, quando a versão deste Driver é atualizada a partir de uma versão anterior à versão **3.1**, esta opção já vem selecionada, mantendo o comportamento das versões anteriores
Add	Adiciona uma operação à lista
Edit	Atualiza a operação selecionada na lista, que equivale a clicar duas vezes em um item
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 deste Driver corresponde ao padrão do protocolo, e esta é 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 em um 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 deste Driver, na **Aba Operations**. Em alguns casos, denominações alternativas frequentes são apresentadas entre parênteses.

Opções disponíveis para tipos de dados

TIPO DE DADOS	FAIXA	DESCRIÇÃO
Char	-128 a 127	Palavra de 8 (oito) bits, caractere. A escrita deve ocorrer sempre em blocos de tamanho par (**Words**)
Byte	0 a 255	Palavra de 8 (oito) bits sem sinal. A escrita deve ocorrer sempre em blocos de tamanho par (**Words**)
Int8	-128 a 127	Palavra de 8 (oito) bits com sinal. A escrita deve ocorrer sempre em blocos de tamanho par (**Words**)
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 ULong)	0 a 4294967295	Inteiro de 32 bits sem sinal (**Double Word**)
Float	-3.4E38 a 3.4E38	Ponto flutuante de 32 bits (**IEEE 754**) (quatro bytes: EXP F2 F1 0). Tratamento para valores não normalizados: **Float Underflow** retorna 0 (zero), **Float Overflow** retorna NULL e **Float NaN** retorna NULL
Float_GE	-1.427E+45 a 1.427E+45	Ponto flutuante de 32 bits usado pela GE, não compatível com **IEEE 754**. É usado em equipamentos GE GEDE UPS, com expoente de 8 (oito) bits $2^{[-128 \ldots +127]}$ e 24 bits de mantissa $[-2^{23} \ldots + (2^{23} - 1)]$. (quatro bytes: EXP F2 F1 F0). Para mais informações, consulte a documentação deste equipamento
Double (ou Real)	-1.7E308 a 1.7E308	Ponto flutuante de 64 bits (**IEEE 754**)
String	Não se aplica	Texto em formato **ANSI**, com número determinado de caracteres ASCII de 8 (oito) bits (**Chars**)
BCD	Consultar descrição	Valor numérico BCD (*Binary-Coded Decimal* ou *Decimal Codificado em Binário*). Ao utilizar este tipo de dados, uma aplicação deve fornecer um valor decimal positivo e inteiro, a ser enviado no formato **BCD**, respeitando o tamanho especificado. O campo **Size**, no caso do tipo de dados **BCD**, refere-se ao número de bytes a serem enviados para representar um valor. Uma vez que na codificação **BCD** cada algarismo é convertido em um *nibble*, tem-se que os valores permitidos devem possuir um número máximo de algarismos igual ao dobro do valor

TIPO DE DADOS	FAIXA	DESCRIÇÃO
		especificado no campo **Size**, ou seja, ao selecionar o valor 2 (dois) para o campo **Size**, o máximo valor que pode ser enviado é 9999. Já se **Size** é igual a 4 (quatro), o valor máximo é 99999999. Os valores permitidos para o campo **Size** no caso de tipos de dados **BCD** são 2 (dois, **Word**) e 4 (quatro, **Double Word**). Para mais detalhes sobre a codificação **BCD**, consulte o tópico **Codificação BCD**
GE_events	Consultar descrição	Tipo de dados utilizado na leitura do *buffer* de eventos (SOE) de um equipamento GE PAC RX7. A definição é permitida apenas em operações que utilizem a função especial de leitura **GE SOE**. Estes eventos são retornados como blocos de 2 (dois) Elementos, com estampas de tempo definidas por um controlador. Para mais informações, consulte o tópico **Leitura de Buffer de Eventos em Controladores GE PAC RX7**
Bit	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 **01**, **02**, **05** e **15**. O campo **Size** não é usado para tipos de dados **Bit**. Quando este tipo de dados é usado, cada Tag ou Elemento de Tag Bloco passa a representar 1 (um) bit
SP_events	Consultar descrição	Tipo de dados utilizado na leitura de eventos (SOE) de relés Schneider Electric das séries SEPAM 20, 40 e 80. Esta definição só é permitida quando uma operação utiliza como função de leitura a função especial **SP SOE**. Estes eventos são retornados como um Bloco de 3 (três) Elementos, com estampa de tempo fornecida por um equipamento. Para mais informações, consulte o tópico **Leitura de Eventos de Relés Schneider Electric SEPAM 20, 40 e 80**
GenTime	1/1/1970 00:00 a 31/12/2035 23:59:59.999 (consulte a **nota** a seguir)	Tipo de dados de data e hora composto por uma estrutura de 8 (oito) bytes, criado originalmente para ser utilizado na leitura de eventos que usam o **algoritmo de SOE Genérico (GenSOE)**. Este tipo de dados pode ser utilizado com as demais funções do protocolo Modbus, além da **GenSOE**. Como este formato é lido

TIPO DE DADOS	FAIXA	DESCRIÇÃO
		internamente como uma estrutura de **Words**, a única função de *swap* válida para este tipo de dados é **Swap Bytes**. A representação deste tipo de dados na memória de um equipamento é descrita no tópico **Tipo de Dados GenTime**. Para mais informações sobre este tipo de dados, consulte o tópico **Algoritmo de Leitura de SOE Genérico da Elipse Software**
Sp_time	1/1/1970 00:00 a 31/12/2035 23:59:59.999 (consulte a **nota** a seguir)	Tipo de dados de data e hora composto por uma estrutura de 8 (oito) bytes, utilizado por relés Schneider Electric das séries SEPAM 20, 40 e 80, geralmente para representar uma estampa de tempo. Para mais informações, consulte a documentação deste equipamento
UTC64d	1/1/1970 00:00 a 31/12/2035 23:59:59.999 (consulte a **nota** a seguir)	Tipo de dados de data e hora representado em formato **Double** (**IEEE 754** 64 bits), com os segundos desde 1/1/1970 00:00
UTC32	1/1/1970 00:00 a 31/12/2035 23:59:59.999 (consulte a **nota** a seguir)	Tipo de dados de data e hora representado em formato inteiro sem sinal de 32 bits, **DWord** ou **UInt**, com os segundos desde 1/1/1970 00:00. Neste formato não são representados os milissegundos, que são considerados sempre como 0 (zero)
Int16_sm	-32767 a 32767	Inteiro de 16 bits com sinal de magnitude
Int32_sm	-2147483647 a 2147483647	Inteiro de 32 bits com sinal de magnitude
Int52 e UInt52	-9007199254740992 a 9007199254740992	Inteiros de 52 bits convertidos para **Double** (ponto flutuante)
Str_I64	"-9223372036854775808" to "9223372036854775807"	Inteiros de 64 bits, com sinal, convertido para o formato **String** (texto)
Str_U64	"0" to "18446744073709551615"	Inteiros de 64 bits, sem sinal, convertido para o formato **String** (texto)

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 as aplicações da **Elipse Software** não possuírem suporte, atualmente, para faixas de valores superiores a este limite em estampas de tempo.
- Os tipos de dados **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.
- Os tipos **Str_I64** e **Str_U64** são manipulados como texto somente no lado de uma aplicação. No lado de um equipamento, estes dados são convertidos automaticamente para o formato numérico por este Driver.

Tipo de Dados GenTime

GenTime é um tipo de dados de data e hora definido e adicionado originalmente a este 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 equipamento, de forma simples.

Em uma aplicação, ou seja, nos valores dos Tags e Elementos 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 deste Driver, é representado por um tipo de dados de data e hora nativo de uma aplicação. Para mais informações sobre os demais tipos de dados de data e hora suportados por este Driver, consulte o tópico **Tipos de Dados Suportados**. Para mais informações sobre os tipos de dados de data e hora de uma aplicação, consulte o respectivo manual do usuário, pois existem algumas diferenças do VBScript utilizado no **Elipse SCADA** para o VBScript utilizado no **Elipse E3**, no **Elipse Power** ou no **Elipse Water**.

No equipamento ou dispositivo **Escravo**, este tipo de dados é representado por uma estrutura composta por 4 (quatro) registradores de 16 bits, ou 8 (oito) bytes, conforme mostrado na tabela a seguir.

Estrutura dos registradores

OFFSET	CONTEÚDO	MAPA DE BITS (16 BITS)	FAIXA (DECIMAL)
0	Ano	AAAAAAAA AAAAAAAA	Entre 0 (zero) e 65535
1	Dia e mês	DDDDDDDD MMMMMMMM	Entre 0 (zero) e 65535
2	Hora e minuto	HHHHHHHH MMMMMMMM	Entre 0 (zero) e 65535
3	Segundo e milissegundo	SSSSSSMM MMMMMMMM	Entre 0 (zero) e 65535

O endereço base, *offset* 0 (zero), a ser atribuído no parâmetro *N4* ou *B4* de um Tag que acessa um dado, contém o ano. O registro seguinte, *offset* 1 (um), tem o dia como o byte mais significativo e o mês como o byte menos significativo. Já no *offset* 2 (dois) tem-se a hora representada no byte mais significativo e os minutos no byte menos significativo. O quarto registro tem os 4 (quatro) bits mais significativos de um **Word** representando os segundos, e os bits restantes, os 2 (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 este Driver a ler um bloco de 4 (quatro) registros Modbus em um equipamento para representar o valor de cada Tag ou Elemento de Bloco para retornar um valor válido.

As vantagens deste tipo de dados são a simplicidade, pois é fácil de gerar no *ladder* de um equipamento, a precisão de milissegundos e a relativa compactação, não necessitando de suporte nativo em um equipamento ou dispositivo **Escravo**.

NOTA

Embora o tipo de dados **GenTime** em si tenha um tamanho de 8 (oito) bytes, ou 4 (quatro) **Words**, a única opção de *swap* que tem efeito sobre este tipo de dados é **Swap Bytes**. Isto porque, conforme visto neste tópico, este tipo de dados é estruturado na memória de um equipamento como tendo 4 (quatro) **Words**, e não é em si mesmo um tipo de dados nativo de um equipamento, e sim deste 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 deste 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 por este 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 por este Driver nas estruturas. Somente não são permitidos os tipos de dados **Bit**, os tipos de dados de 8 (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 definido um tipo de dados, o usuário pode associá-lo a qualquer Tag, desde que 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 função lê apenas bits.

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

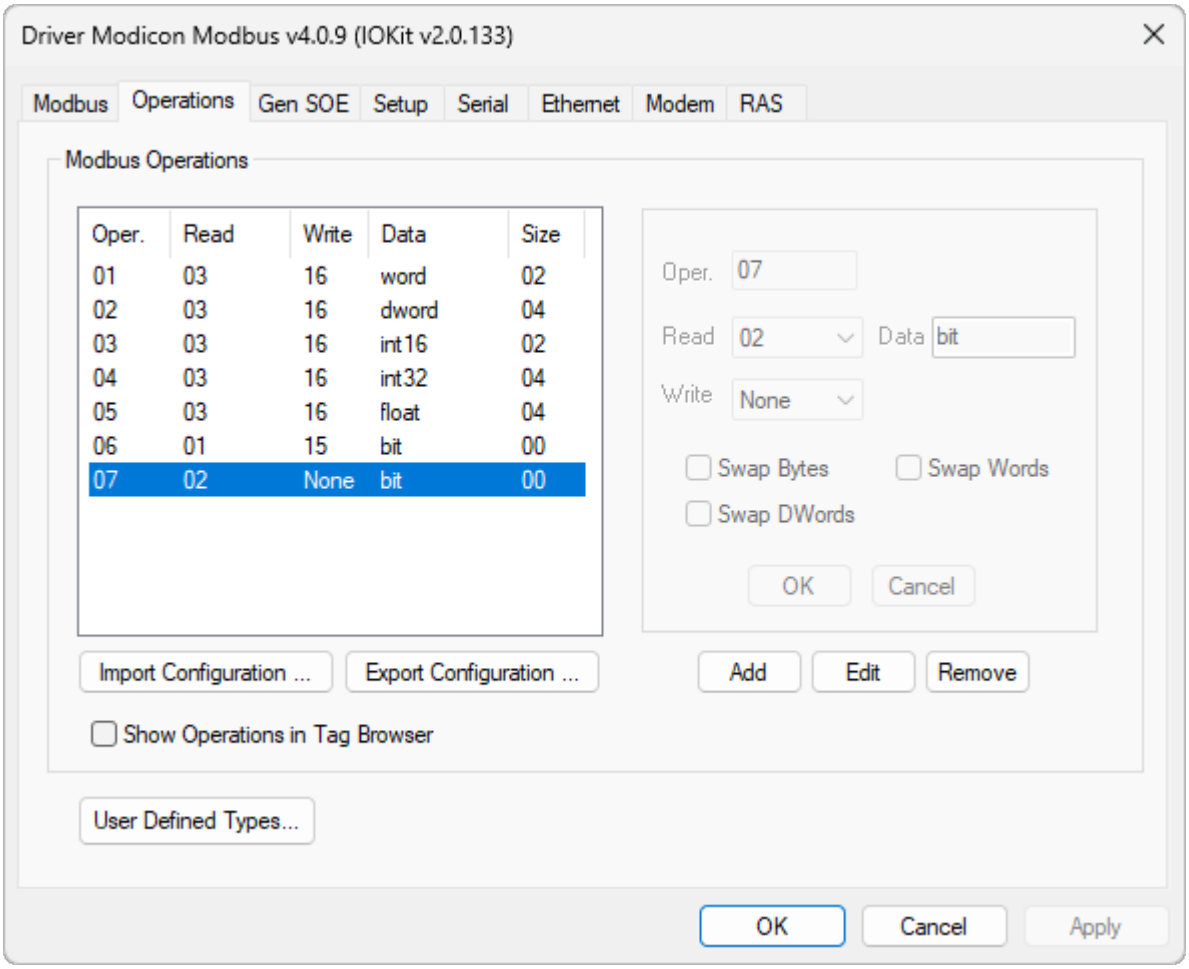
Aplicações

Os tipos de dados definidos pelo usuário foram originalmente implementados para uso em conjunto com a **rotina de SOE genérico deste Driver** (**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 um equipamento em uso por algum motivo não permita o uso de Superblocos. Para mais informações, consulte o tópico **Leitura por Superblocos (Agrupamento)**.

Configuração de Tipos de Dados Definidos pelo Usuário

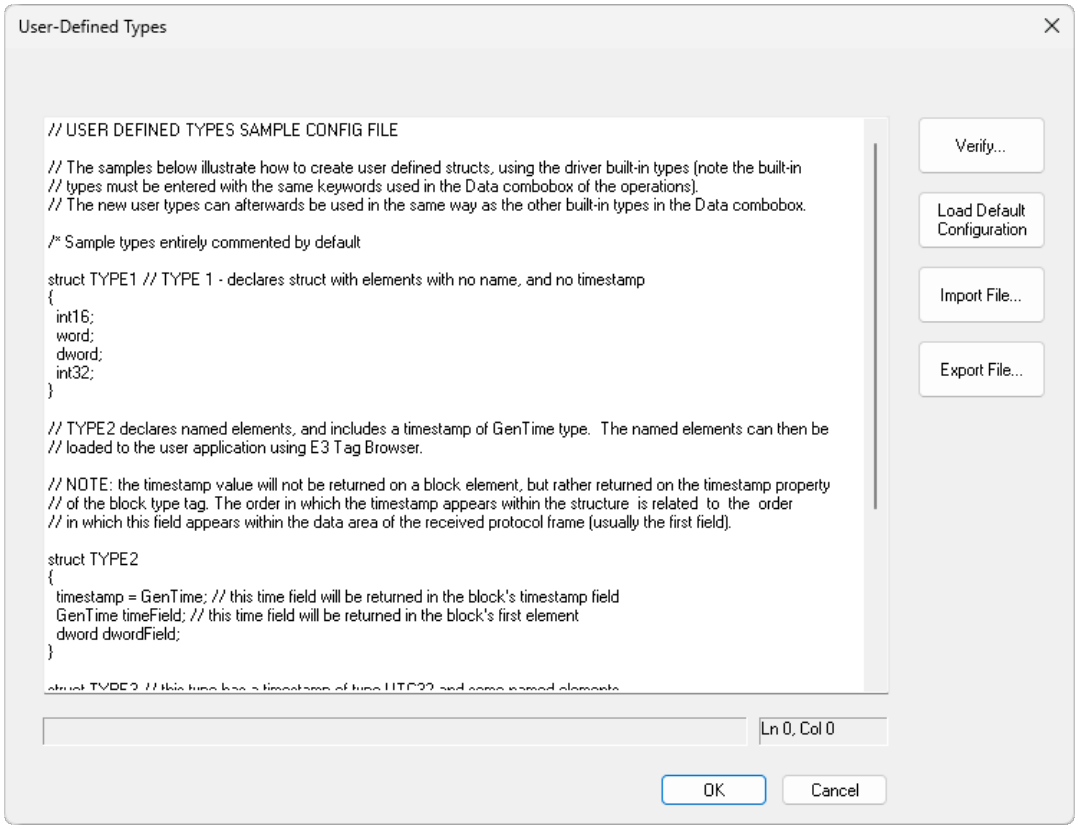
A configuração de tipos de dados definidos pelo usuário é realizada em um janela específica, clicando em **User Defined Types** na aba **Operations** da janela de configurações deste Driver, conforme a figura a seguir.



Aba Operations da janela de configurações deste 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 3 (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 3 (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 da linguagem C++, iniciando por "/*" e terminando por "*/". Note que o arquivo de exemplo que vem com este Driver já aplica este formato de comentário aos 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 3 (três) tipos de dados de exemplos estejam prontos para uso.

À medida que o texto do arquivo de configuração é alterado, a barra de status mostra o resultado da análise sintática deste arquivo, em tempo real. Esta barra de status mostra a mensagem "Status: OK!" se não são detectados erros neste 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 o número da linha e da coluna onde este erro foi detectado.

A verificação pode ser realizada 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 de dados tem a sintaxe a seguir, em que 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];
}
```

Em que:

- **struct**: Palavra-chave, em letras minúsculas, que inicia a definição do tipo de dados definido pelo usuário
- **<Nome do Tipo>**: Nome pelo qual o tipo de dados é identificado por este Driver. Este é o nome mostrado na opção **Data**, na configuração das operações. Deve ter no máximo 6 (seis) caracteres
- **timestamp**: Campo opcional que indica que a estrutura tem uma estampa de tempo definida pelo equipamento, que deve ser retornado na propriedade **Timestamp** de um Tag. Cada estrutura pode ter no máximo 1 (uma) estampa de tempo. A ordem em que aparece na estrutura influencia a posição em que um campo é lido no *frame* retornado por um equipamento. Note que nos Tags este valor é retornado somente na propriedade **Timestamp**. Podem ser definidos quaisquer tipos de dados de data e hora suportados por este Driver. Na versão atual, este 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 na janela 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 formato, é preciso preceder um número com o prefixo "0x", como por exemplo "0x10" para codificar o valor decimal 16 em hexadecimal
- **<tipo data e hora>**: Tipos de dados de data e hora pré-definidos por este Driver, que podem ser utilizados como estampa de tempo por um dispositivo **Escravo**. Na atual versão deste Driver, são aceitos os tipos de dados nativos **GenTime**, **Sp_time**, **UTC32** e **UTC64d**
- **<tipo>**: Tipo de dados de um elemento. Deve ser definido como um dos tipos de dados pré-definidos por este Driver, e escrito da mesma forma que aparece na opção **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 8 (oito) bits e nem tipos de dados de tamanho variável, tais como **BCD** e **String**
- **[nome do elemento]**: Parâmetro opcional que define o nome de cada Elemento de um Tag Bloco. Se definido, determina o nome dos Elementos de um Tag Bloco nos Tags presentes na janela Tag Browser do **Elipse E3**, do **Elipse Power** ou do **Elipse Water**. Se não é definido na declaração da estrutura, este Driver atribui nomes padrão aos Elementos na janela Tag Browser, com a palavra-chave "Element" seguida do valor do índice de um Elemento dentro de um Tag Bloco, tais como "Element1", "Element2", e assim por diante

Importação e Exportação

As opções **Import File** e **Export File** permitem importar e exportar um 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 de um arquivo, ou para compartilhá-lo entre vários Drivers. Este arquivo é sempre gravado e lido no formato **ANSI** padrão do Windows, usando o conjunto de caracteres **Windows-1252**. Futuras versões deste Driver podem suportar outros formatos.

Além de copiar este 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 deste arquivo em outro Editor de Texto.

A opção **Load Default Configuration** carrega novamente no editor o arquivo padrão de configuração, 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 neste arquivo são descartadas por este Driver. Clicando-se em **OK**, este arquivo é armazenado em uma aplicação. Esta operação realiza a verificação completa deste arquivo e, ao identificar algum erro, este é mostrado e a janela não é fechada. Em caso de necessidade de salvar alterações com erros ainda pendentes, exporte este 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 deste Driver, desde que este nome tenha sido previamente declarado, como explicado anteriormente neste tópico.

IMPORTANTE

Como no **Elipse E3**, no **Elipse Power** ou no **Elipse Water** 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 estes 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 de um equipamento com *Id* 5 (cinco), interpretado como um tipo de dados definido pelo usuário chamado "mytype", com *Slave Id* no campo **Item**:
  - **Dispositivo**: "" (**String** vazia)
  - **Item**: "5:shr100.mytype"

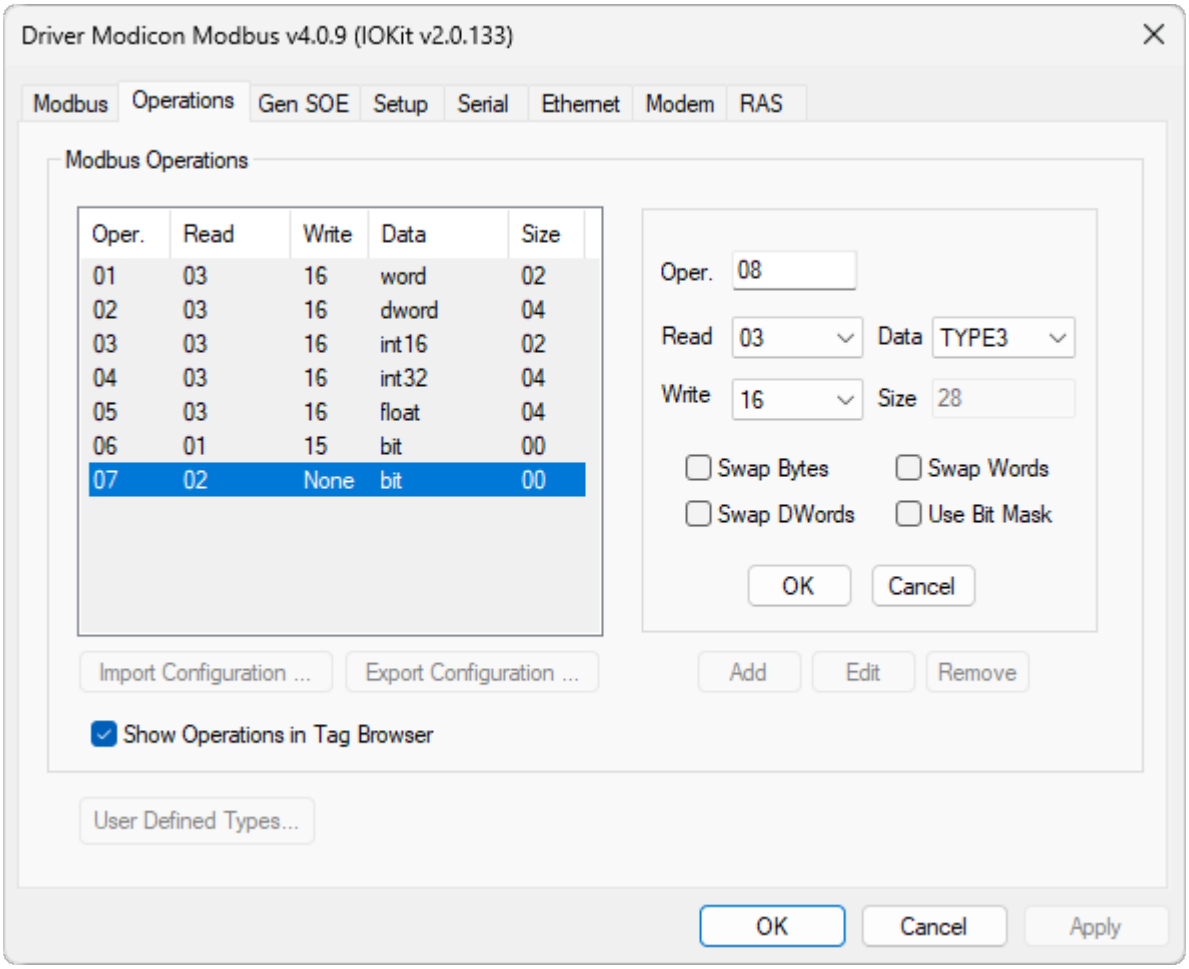
NOTA

As opções de permuta, ou 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 os **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 em um arquivo de configuração na janela **User-Defined Types**, estes tipos de dados estão disponíveis para uso nas operações deste 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 operação que utiliza o tipo de dados definido pelo usuário, ou estrutura, de nome **TYPE3**, mostrado no exemplo acima, após o usuário clicar em **Add**.

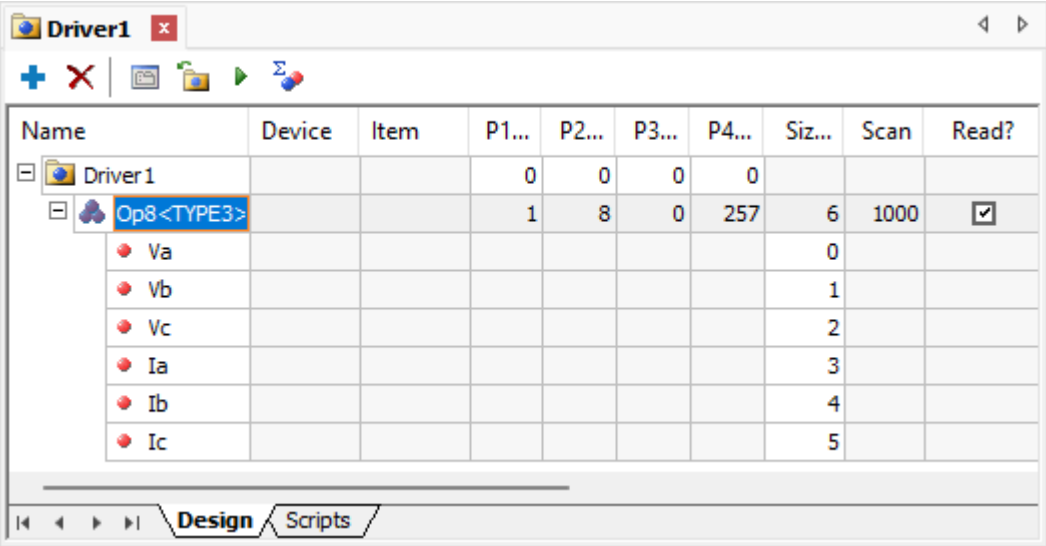


Adicionar um tipo de dados definido pelo usuário

NOTA

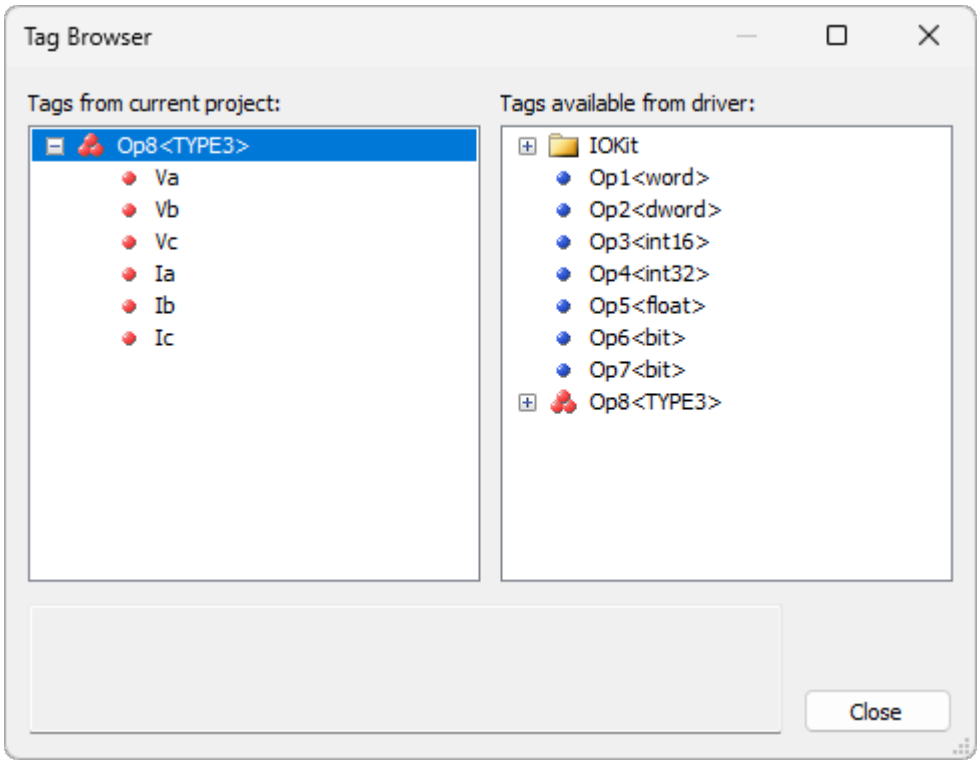
As opções de permuta para tipos de dados definidos pelo usuário têm efeito apenas nos elementos da estrutura definida e não na estrutura inteira, ou seja, se a opção **Swap Words** está habilitada, todos os elementos com mais de 16 bits têm os **Words** permutados. Os elementos de 16 bits, entretanto,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

Após a definição de uma operação usando o tipo de dados **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 Elipse E3, no Elipse Power ou no Elipse Water

Se o nome para cada elemento desta estrutura foi definido, é possível utilizar a janela Tag Browser do **Elipse E3**, do **Elipse Power** ou do **Elipse Water** para incluir um Tag Bloco referente ao tipo de dados desejado em uma 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 este procedimento.



Uso da janela Tag Browser para definir Tags usando estruturas

Como a figura sugere, clique em na aba **Design** deste Driver para abrir a janela Tag Browser e arraste o tipo de dados 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 de um equipamento, e podem ser usados com o **algoritmo de leitura de SOE genérico da Elipse Software**. Caso seja necessário ler eventos organizados na memória de um equipamento, como uma sequência de estruturas, em

uma operação que utilize apenas a **função pública de leitura do protocolo**, ou seja, sem o emprego de funções especiais com algoritmo de SOE, este procedimento pode ser realizado das seguintes formas:

- **Leitura em Bloco**: Crie um Tag 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 2 (dois) elementos que represente eventos acumulados em um arranjo na memória de um equipamento. Caso se deseje ler em bloco 5 (cinco) eventos, defina 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** de um Tag para ler um bloco de dados. Com isto, considerando o exemplo do item anterior, ao invés de criar um Tag Bloco com 10 Elementos, o usuário precisa criar apenas um único Tag Bloco com 2 (dois) Elementos, configurando o parâmetro *B3* com o valor "5". Desta forma, ao realizar a leitura deste Tag, o **Elipse E3**, o **Elipse Power** ou o **Elipse Water** executa 5 (cinco) vezes o evento **OnRead** deste Tag, e em cada execução os Elementos e propriedades deste 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 a um Tag, no evento **OnRead** do Tag reportado a evento. Para mais informações, consulte o tópico sobre Tags Reportados a Eventos no *Manual do Usuário do Elipse E3*

Em outras palavras, todo Tag de Comunicação que use estruturas e que utilize uma **função pública de leitura do protocolo**, pois este recurso não funciona para **funções especiais de SOE**, torna-se um Tag Reportado a Eventos se o parâmetro *B3* é 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 de uma função.

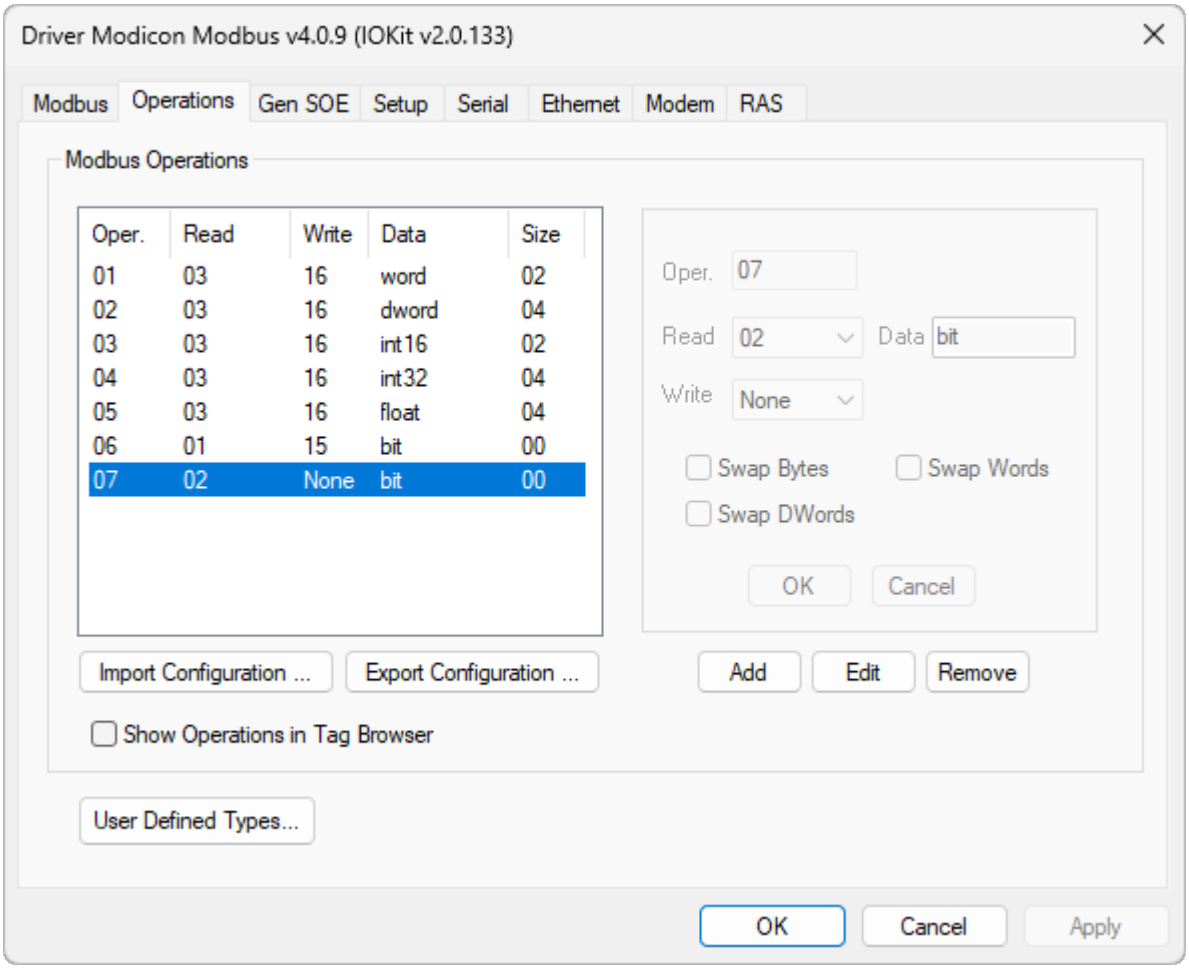
Para mais informações a respeito da configuração de Tags de Comunicação, consulte o tópico **Configurando um Tag de Comunicação**.

IMPORTANTE

Ao ler eventos de memória de massa em Tags reportados a eventos no **Elipse E3**, no **Elipse Power** ou no **Elipse Water**, desabilite a banda morta neste Tag, ou seja, configure a propriedade **EnableDeadBand** com o valor Falso, e também no objeto Histórico associado, ou seja, configure a propriedade **DeadBand** com o valor 0 (zero), para evitar a perda de eventos com valores próximos. Também é importante desabilitar o histórico por varredura, ou seja, configure a propriedade **ScanTime** com o valor 0 (zero) no **Elipse E3**, no **Elipse Power** ou no **Elipse Water**. Com isto, garante-se que novos eventos só sejam armazenados através do método **WriteRecord**, executado no evento **OnRead** de um Tag, evitando a duplicação de eventos.

Importação e Exportação de Operações

A importação e exportação de operações deve ser realizada na aba **Operations** deste 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 no formato **INI**.

Nas versões anteriores à versão **2.0** deste Driver, a configuração das operações era executada no arquivo modbus.ini, carregado no momento da inicialização de um objeto Driver. Os arquivos modbus.ini destas versões antigas ainda podem ser carregados na versão atual deste Driver, utilizando a opção de importação.

NOTA

As operações deste 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

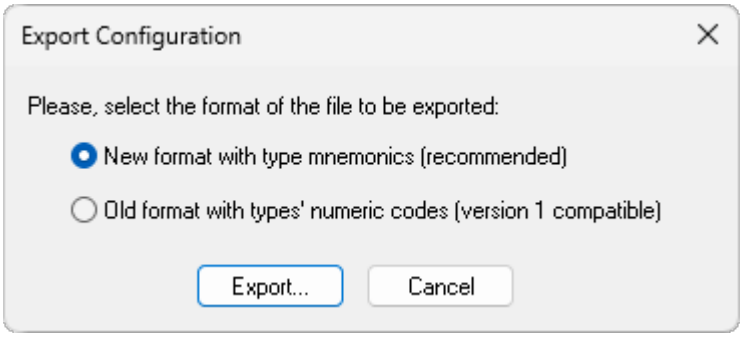
A importação de arquivos de configuração é bastante simples. Clique em **Import Configuration** e selecione um arquivo no formato **INI**. Este Driver deve carregar as configurações de operações, que aparecem imediatamente no quadro **Modbus Operations**. Este Driver permite a importação de arquivos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

Exportação

A exportação de arquivos de configuração de operações pode ser realizada para compartilhar a mesma configuração de operações em outros 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 deste 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 a seguir.

Ao clicar em **Export Configuration**, abre-se uma janela com as opções mostradas na figura a seguir.



Opções de exportação

Nesta janela deve-se selecionar entre exportar conforme o novo padrão, opção **New format with type mnemonics**, com os tipos de dados exibidos definidos com **Strings** (mnemônicos), ou no formato antigo, opção **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 opção **Data** na aba **Operations**.

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

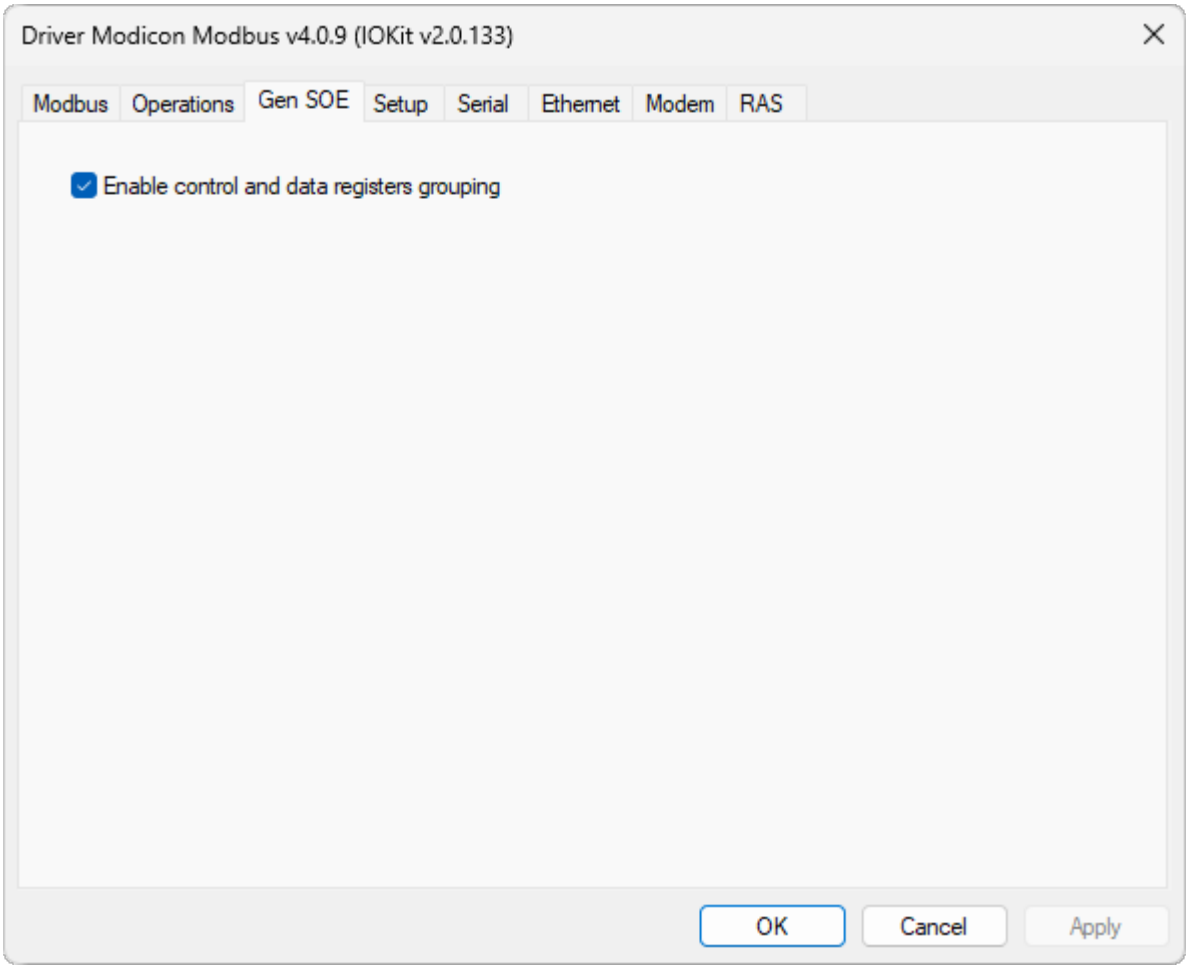
Já o formato antigo deve ser selecionado somente no caso de ser imprescindível exportar para versões anteriores à versão **2.8** deste Driver.

Note que, para exportar com sucesso arquivos modbus.ini a serem carregados em versões anteriores à versão **2.0**, 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 deste 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 é **Enable control and data registers grouping**, que habilita o agrupamento de registros de controle e registros de dados, de forma a realizar o mínimo de leituras possível. Se esta opção está habilitada (padrão), este 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. Este 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 equipamento. Os equipamentos ATOS não permitem a leitura agrupada dos registros de controle e de estruturas de dados, exigindo que esta opção esteja desabilitada.

Configuração em Modo Offline

As configurações deste Driver também podem ser acessadas em tempo de execução se este Driver é 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
ModiconModbus.ModbusMode	Inteiro. Os valores possíveis são **0**: Modbus RTU, **1**: Modbus ASCII ou **2**: Modbus TCP
ModiconModbus.Olderaddr	Booleano. Os valores possíveis são **0**: Dado endereçado a partir de 0 (zero) ou **1**: Dado endereçado a partir de 1 (um)
ModiconModbus.UseDefaultSlaveAddress	Booleano. Os valores possíveis são 0 (zero) ou 1 (um)

PARÂMETRO	TIPO DE DADOS
ModiconModbus.DefaultSlaveAddress	Inteiro sem sinal
ModiconModbus.UseSwapAddressDelay	Booleano. Os valores possíveis são 0 (zero) ou 1 (um)
ModiconModbus.SwapAddressDelay	Inteiro, com o intervalo de *delay* em milissegundos. **NOTA**: Opção obsoleta e mantida por compatibilidade. Para aplicações novas, use a opção **Inter-frame delay** da aba **Serial** da biblioteca **IOKit**
ModiconModbus.WaitSilenceOnError	Booleano. Os valores possíveis são 0 (zero) ou 1 (um)
ModiconModbus.UseGatewayTCP	Booleano. Os valores possíveis são 0 (zero) ou 1 (um)
ModiconModbus.EnableCMSAddressing	Booleano. Os valores possíveis são 0 (zero) ou 1 (um)
ModiconModbus.ConfigFile	**String** contendo o arquivo de configuração com as operações deste Driver. Este arquivo pode ser exportado e importado na aba **Operations** da janela de configurações deste Driver
ModiconModbus.EnableReconnectAfterTimeout	Booleano. Os valores possíveis são **0**: O *time-out* não gera desconexão do meio físico ou **1**: Em caso de *time-out*, quando em meio físico Ethernet, este Driver promove a desconexão e reconexão do meio físico. Para comunicação do protocolo serial **Modbus RTU/ASC** via Ethernet, a reconexão sempre é efetuada após um *time-out*
ModiconModbus.UseDemotion	Booleano. Os valores possíveis são **0**: Desabilita o modo **Auto-Demotion** ou **1**: Habilita o modo **Auto-Demotion**
ModiconModbus.DemotionTime	Inteiro. Define o tempo de inatividade, em segundos, para um equipamento **Escravo** que entrou em modo **Auto-Demotion**
ModiconModbus.DemotionErrorCount	Inteiro. Define a quantidade limite de falhas consecutivas por *time-out*, para colocar um equipamento **Escravo** em modo **Auto-Demotion**
ModiconModbus.UserTypesConfigFile	**String** de configuração dos **tipos de dados definidos pelo usuário**, ou estruturas. Trata-se do mesmo arquivo de configuração que pode ser acessado na janela de configuração deste Driver, na opção **User-Defined Data Types**
ModiconModbus.EnableGenSOERegGrouping	Booleano. Os valores possíveis são **0**: O algoritmo de leitura de eventos primeiro lê os registros de controle e em seguida os dados de eventos ou **1**: 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
ModiconModbus.ShowOperationsInTagBrowser	Booleano. Os valores possíveis são **0**: A janela Tag Browser mostra os modelos de Tags configurados por **Strings**, que é o comportamento padrão, ou **1**: A janela Tag Browser mostra os modelos de Tags configurados numericamente, que é o comportamento legado

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 nas categorias **Configurando um Tag de Comunicação** e **Configurando Tags Especiais**.

Configurando um Tag de Comunicação

Os Tags de Comunicação deste Driver são os Tags que permitem a comunicação com 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 distingue 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 de um 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 mais informações sobre funções, consulte a especificação no *site oficial do protocolo*.

Estes Tags podem ser configurados das seguintes formas:

- **Configuração por Strings**: Este é o método mais novo que pode ser utilizado com o **Elipse E3**, o **Elipse Power** ou o **Elipse Water** 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**, por 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 de aplicações.

Leitura em Bloco

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

Campo Dispositivo

No campo **Dispositivo**, o *Slave Id*, endereço identificador de um equipamento, deve ser fornecido como um número entre 0 (zero) e 255 seguido de dois pontos, como por exemplo "1:", "101:" ou "225:".

Cabe lembrar que, no protocolo Modbus RTU Serial, o *Slave Id* "0" (zero) é reservado para *broadcast*, o que só faz sentido ser usado para operações de escrita, pois não há retorno dos equipamentos, ou ocorrem 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 está detalhada a seguir.

Campo Item

No campo **Item** devem ser fornecidas todas as demais informações de endereçamento e da operação a ser realizada por um Tag, com a sintaxe a seguir, em que os parâmetros entre colchetes são opcionais.

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

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

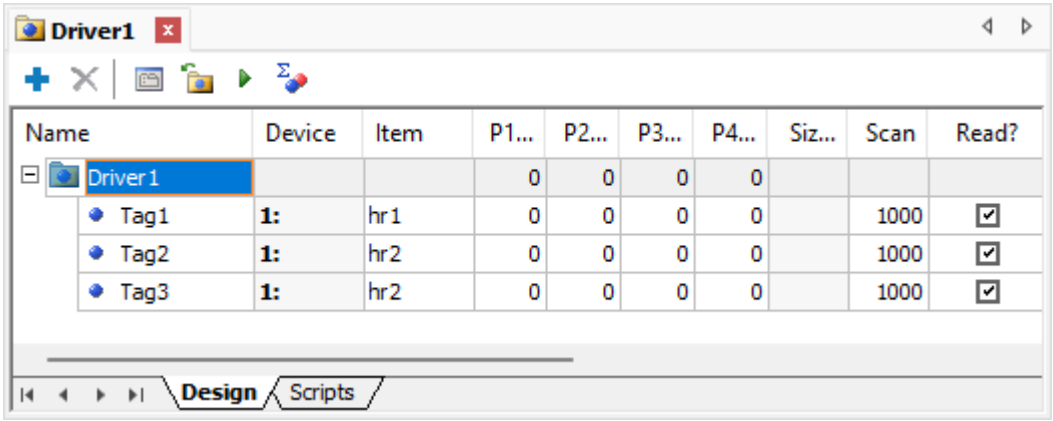
```
<slave id>:<espaço de endereçamento><endereço>[.<tipo>[<tam. do tipo>]][.<byte order>][/bit]
```

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 em uma aplicação.

1. Leitura ou escrita de *Holding Registers*, funções **03** e **16**, de endereço 100 de um equipamento com *Slave Id* 1 (um) e com *Slave Id* no campo **Dispositivo**:
   a. **Dispositivo**: "1:"
   b. **Item**: "hr100"
2. Leitura ou escrita de *Holding Registers*, funções **03** e **16**, de endereço 120 de um equipamento com *Slave Id* 3 (três) e com *Slave Id* no campo **Item**:
   a. **Dispositivo**: "" (**String** vazia)
   b. **Item**: "3:hr120"
3. Leitura ou escrita de *Coils*, funções **01** e **15**, de endereço 65535 (FFFFh) de um equipamento com *Slave Id* 4 (quatro), 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 **Elipse E3**, do **Elipse Power** ou do **Elipse Water** configurados por **Strings**.



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. Para mais informações, consulte a **tabela** em seção específica, mais adiante neste mesmo tópico.

2. **Endereço**: Valor numérico identificando o endereço de um 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", como por exemplo "&hFFFF". Já para valores em octal, use o prefixo "&o", como por exemplo "&o843". Este endereço pode ter um deslocamento (*offset*) em relação ao endereço realmente enviado em um *frame* de comunicação, o que depende da convenção utilizada pelo fabricante. Em caso de dúvida sobre as convenções de endereçamento, consulte o tópico **Dicas de Endereçamento (Modbus Convention)**. Em especial, verifique se um equipamento implementa o *offset* padrão do *Data Model* do protocolo. Consulte 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, e são também melhor detalhados mais adiante:

1. **Tipo**: Define como os bytes da área de dados de um *frame* de comunicação devem ser interpretados. Se omitido, são usados os tipos de dados 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 de dados suportados, mais adiante neste tópico.

2. **Tamanho do tipo**: É necessário especificar este campo apenas para os tipos de dados de tamanho variável, como **BCD** e **String**. O valor numérico indica o tamanho do tipo de dados 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 em um *frame* de comunicação, o chamado *big endian*. Consulte 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 ou **Words**. Em geral recomenda-se não usar este recurso, dando preferência ao mapeamento de bits de uma aplicação. 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 de dados. O tamanho máximo atualmente é 64 para tipos de dados **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 no 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 os respectivos tipos de dados nativos.

Espaços de endereçamento e mnemônicos

ESPAÇO DE ENDEREÇAMENTO	MNEMÔNICO	TIPO DE DADOS NATIVO	FUNÇÃO	COMENTÁRIOS
Holding Register	hr	**Word** (16 bits)	Funções **03** e **16**	As funções **03** e **16** são as mais usadas do protocolo (Modbus classe 0)
Single Holding Register	shr	**Word** (16 bits)	Funções **03** e **06**	A função **06** escreve nos mesmos registradores da função **16**, a diferença é que a função **16** pode escrever em Blocos, enquanto a função **06** escreve em apenas um registro por vez, mas com menor *overhead*
Coil	cl	**Bit**	Funções **01** e **15**	
Single Coil	scl	**Bit**	Funções **01** e **05**	A função **05** escreve nos mesmos registros da função **15**, porém não pode escrever em Blocos, portanto com menor *overhead*
Discrete Input	di	**Bit**	Funções **02** e **None**, somente leitura	
Input Register	ir	**Word** (16 bits)	Funções **04** e **None**, somente leitura	
Exception Status	es	**Byte**	Funções **07** e **None**, somente leitura	
File Register	fr	**Word** (16 bits)	Funções **20** e **21**	
ABB MGE 144 - Leitura de Memória	abbmge	**Word** (16 bits)	Funções **65 03** e **None**, somente leitura	

ESPAÇO DE ENDEREÇAMENTO	MNEMÔNICO	TIPO DE DADOS NATIVO	FUNÇÃO	COMENTÁRIOS
de Massa				
ABB MGE 144 - Reset	abbmge.rst	Não usado	Função de leitura **None** e de escrita **65 01**	
ABB MGE 144 - Zera Memória de Máximos e Mínimos	abbmge.rstmxmn	Não usado	Função de leitura **None** e de escrita **65 02**	
GE PAC RX7 SOE - Leitura	gesoe	**GE_events**	Função de leitura **GE SOE** e de escrita **None**	
SEPAM SOE Events	spsoe	**SP_events**	Função de leitura **SP SOE** e de escrita **None**	Coleta de um medidor e retorna a um Tag uma estrutura, do tipo **SP_events**, com eventos de quaisquer pontos. Para mais informações, consulte o tópico **SEPAM SOE**
SEPAM SOE Single Point Events	ptspsoe	**Int32**	Função de leitura **SP SOE** e de escrita **None**	Coleta de um medidor e retorna a um Tag um valor inteiro do campo **Edge**, relativo a eventos de um ponto específico. Para mais informações, consulte o tópico **SEPAM SOE**
Elipse Generic SOE	elsoe	**Word** (16 bits)	Função de leitura **Gen SOE**, função Modbus **03** com algoritmos adicionais, e de escrita **16**	

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 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 deste Driver, e também um *alias* ou nome alternativo.

Tipos de dados suportados

TIPO	MNEMÔNICO	ALIAS
Char	char	ch
Byte	byte	by ou u8
Int8	int8	i8
Int16	int16	i16
Int32	int32	i32
Word ou UInt	word	u16
DWord ou ULong	dword	u32
Float	float	f
Float_GE	float_GE	fge
Double ou Real	double	d
Int52	int52	i52
UInt52	uint52	u52
String	string	s
Str_I64	str_i64	s_i64
Str_U64	str_u64	s_u64
BCD	BCD	bcd
GenTime	GenTime	gtm
Sp_time	Sp_time	sptm
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. Para mais informações, consulte 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, este Driver não permite o uso destes tipos de dados no campo **Item**. Para mais informações, consulte o tópico **Tipos de Dados Definidos pelo Usuário**.

Exemplos

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

   a. **Dispositivo**: "1:"

   b. **Item**: "hr100.u32" ou "hr100.dword", ou se é conveniente usar hexadecimal, "hr&h64.u32"

2. Leitura ou escrita de *Holding Registers*, funções **03** e **16**, de endereço 150 de um equipamento com *Slave Id* 3 (três), interpretado como um **Float**, com *Slave Id* no campo **Item**:

a. **Dispositivo**: "" (**String** vazia)

b. **Item**: "3:hr150.f" ou "3:hr150.float" ou em hexadecimal "3:hr&h96.f"

3. Leitura ou escrita de *Holding Registers*, funções **03** e **16**, de endereço 1500 de um equipamento com *Slave Id* 5 (cinco), interpretado como um **Double**, com *Slave Id* no campo **Item**:

a. **Dispositivo**: "" (**String** vazia)

b. **Item**: "5:hr1500.d" ou "5:hr1500.double" ou ainda, se é conveniente endereçar em hexadecimal, "5:hr&h5DC.d"

4. Leitura ou escrita de *Holding Registers*, funções **03** e **06**, de endereço 100 de um equipamento com *Slave Id* 5 (cinco), interpretado como um **Word**, com *Slave Id* no campo **Item**:

a. **Dispositivo**: "" (**String** vazia)

b. **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

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

a. **Dispositivo**: "" (**String** vazia)

b. **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 de dados **Double** exige a leitura de 4 (quatro) endereços de *Holding Registers*. Pelos mesmos motivos, a leitura e escrita de Tags de "hr" de tipo de dados **Byte** só pode ser realizada aos pares. Em um equipamento, cada endereço de *Holding Register* contém sempre 2 (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, ou *type size*, em bytes, logo após o tipo de dados.

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

1. Leitura ou escrita de *Holding Registers*, funções **03** e **16**, de endereço 100 de um equipamento com *Slave Id* 1 (um), interpretado como uma **String** de 10 bytes, ou 5 (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 de um equipamento com *Slave Id* 1 (um), interpretado como **BCD** de 8 (oito) dígitos, ou 4 (quatro) bytes ou 2 (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 um 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 um 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 este Driver lê um valor igual a "1234h", ou "4660" em decimal, por padrão espera que este dado seja enviado com a sequência de bytes 12h e 34h. Se entretanto um equipamento usa o padrão inverso, chamado de *little endian*, é enviado primeiro o byte 34h e depois o 12h, e este Driver pode interpretá-lo como 3412h, ou 13330 em decimal, a não ser que os 2 (dois) bytes tenham a ordem invertida antes da interpretação.

No caso de valores de 32 bits pode também ocorrer de valores em **Word** permutados, ou *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, este 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". Para mais informações, consulte a tabela a seguir.

Se o parâmetro *byte order* é omitido, um 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
b0				msb	-	by7 by6 by5 by4 by3 by2 by1 by0
b1	X			-	sb	by6 by7 by4 by5 by2 by3 by0 by1
b2		X		-	sw	by5 by4 by7 by6 by1 by0 by3 by2
b3	X	X		-	sb.sw	by4 by5 by6 by7 by0 by1 by2 by3
b4			X	-	sdw	by3 by2 by1 by0 by7 by6 by5 by4
b5	X		X	-	sb.sdw	by2 by3 by0 by1 by6 by7

	SWAP BYTES	SWAP WORDS	SWAP DWORDS	ALIAS	ALIAS 2 (SWAPS)	ORDENAMENTO
						by4 by5
b6		X	X	-	sw.sdw	by1 by0 by3 by2 by5 by4 by7 by6
b7	X	X	X	lsb	sb.sw.sdw	by0 by1 by2 by3 by4 by5 by6 by7

NOTA

A coluna **Ordenamento** corresponde a 64 bits, em que **b0** é o byte menos significativo (**lsb** ou *least significant byte*) e **b7** é o byte mais significativo (**msb** ou *most significant byte*).

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

Já o ordenamento "b1" realiza a permuta dos bytes, dois a dois, ou seja, ao receber um **Word**, um 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 ordenamento "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, ao receber de um equipamento o valor 01020304h, o valor utilizado para os Tags da aplicação é 03040102h.

Já ao usar o ordenamento "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 ordenamento "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 colunas **Alias** e **Alias 2 (swaps)** 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 de um equipamento especifica qual o ordenamento utilizado, ou como configurá-lo, pois 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 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 as situações a seguir:

1. Caso um 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 um 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 por um 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 de dados com tamanho de oito bits, tais como **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 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 de um equipamento com *Slave Id* 5 (cinco), 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 de um equipamento com *Slave Id* 5 (cinco), 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 de um equipamento com *Slave Id* 5 (cinco), 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 deste Driver

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

Tags especiais

DEVICE	ITEM	OPERAÇÃO
	ForceWaitSilence	Somente escrita
<slave id>:	**LastExceptionCode**	Leitura ou escrita
<slave id>:	**DemotionStatus**	Somente leitura

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**, no **Elipse Power** ou no **Elipse Water**, 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**, o **Elipse Power** ou o **Elipse Water**, 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**, este 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** neste Driver.

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

As operações com tipos de dados de 8 (oito) bits, como o tipo de dados **Byte**, implicam sempre, obviamente, na leitura de pelo menos 2 (dois) bytes, um registro Modbus de 16 bits. Para prevenir surpresas para o usuário, este Driver exige que a escrita de dados de 8 (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* ou *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 ou B1**: Endereço de um equipamento **Escravo** (CLP) na rede (*Slave Id*). Este endereço pode variar de 1 (um) a 247 para redes seriais via Modbus RTU ou variar de 0 (zero) a 255 para comunicação direta com equipamentos Ethernet via Modbus TCP. Para o modo **Serial**, o identificador com valor 0 (zero) é reservado para o modo **Broadcast**, enviando a mensagem para todos os equipamentos **Escravos** (CLP) que estiverem na rede. Em **Ethernet**, no 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 RS-485, com Modbus RTU, usando um *gateway* capaz de realizar a conversão de **Modbus TCP** para **Modbus RTU**.

NOTA

No Modbus **Serial**, o modo **Broadcast** com o parâmetro *N1* igual a 0 (zero) não permite efetuar 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 ou B2**: Código da operação. Referencia uma operação adicionada na janela de configurações deste Driver. Para mais informações, consulte o tópico **Aba Operations**.
- **N3 ou B3**: Parâmetro adicional. Este parâmetro em geral não é usado e pode ser deixado em 0 (zero). Só é usado nas seguintes situações:
  - **Funções Modbus 20 e 21**: No caso de operações que utilizem estas funções de acesso a arquivo, as funções **20 e 21**, o parâmetro *N3* ou *B3* especifica o arquivo a ser acessado.

- **Use Bit Mask**: Para Tags referenciando operações com a opção **Use Bit Mask** habilitada, o parâmetro *N3* ou *B3* especifica o número do bit a ser acessado. Para mais informações, consulte o tópico **Aba Operations**.

- **Tipos de Dados Definidos pelo Usuário**: Para operações que usem estruturas, se o parâmetro *B3* é maior que 0 (zero), define o retorno de um vetor de blocos reportado a eventos, por meio de uma sequência de eventos **OnRead** de um Tag. Para mais informações, consulte 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* ou *B3* indica o tamanho da tabela associada na memória de um equipamento ou dispositivo **Escravo**, em número máximo de eventos comportados. Para mais informações, consulte o tópico **Algoritmo de Leitura de SOE Genérico da Elipse Software**.

- **N4 ou B4**: Endereço do registrador, variável ou bit em um equipamento ou dispositivo **Escravo** (CLP) em que se deseja ler ou escrever, conforme o mapa de registradores deste equipamento. Para mais informações, consulte o manual deste equipamento. É importante ajustar corretamente a opção **Data Address Model Offset**. Para mais informações, consulte o tópico **Aba Modbus** e verifique se a documentação do fabricante não usa *offsets* utilizados por equipamentos Modbus antigos, da chamada **Modbus Convention**.

- **Tamanho ou Índice (somente para Tags Bloco)**: Cada Elemento de Bloco representa o valor de um dado com tipo de dados definido na operação utilizada, no parâmetro *N2* ou *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, este Driver necessita ler 2 (dois) registros consecutivos de um equipamento.

Tags Especiais

Além dos Tags de Comunicação, ou seja, Tags que referenciam operações, existem também Tags especiais para executar funções específicas deste Driver. Estes Tags estão descritos no tópico **Configurando Tags Especiais**.

Dicas de Endereçamento (Modbus Convention)

No tópico **Configurando um Tag de Comunicação**, o endereçamento dos Tags, ou seja, os parâmetros *N4* ou *B4* na **configuração numérica**, é descrito com base na especificação mais recente do protocolo Modbus, a versão **1.1b**. Entretanto, há equipamentos que ainda utilizam a antiga convenção de endereçamentos com *offsets* conhecida como **Modbus Convention**, que acrescenta *offsets* a um endereço. Este tópico explica como endereçar os Tags caso o mapa de registradores de um 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 a um equipamento, acrescido ou não do *offset* padrão de 1 (um), requerido pelo Modbus Data Model especificado pelo protocolo, conforme configuração da opção **Data Model Offset** na aba **Modbus** da janela de configurações deste 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 a um endereço, cujo valor depende da função Modbus utilizada, ou mais especificamente, de qual espaço de endereçamento, ou *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), **05**: Write Single Coil (0x) e **15**: Write Multiple Coils (0x)	00000
Discrete Inputs	**02**: Read Discrete Inputs (1x)	10000
Input Registers	**04**: Read Input Registers (3x)	30000

TIPO DE DADOS (STANDARD DATA MODEL)	FUNÇÃO MODBUS	OFFSET
Holding Registers	**03**: Read Holding Registers (4x), **06**: Write Single Register (4x) e **16**: Write Multiple Registers (4x)	40000
File Register (antiga Extended Memory file)	**20**: Read General Reference (6x) e **21**: Write General Reference (6x)	60000

Se o mapa de registradores de um 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 de um 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 no endereço.

Exemplos

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

As funções Modbus **20** e **21** tem como característica manipular leitura ou escrita de múltiplos sub-grupos de dados, alocadas na memória extendida do equipamento, ou referências 6XXXXX, conforme a tabela a seguir.

Memória extendida ou referências 6XXXXX

MEMORY FILE NUMBER 1		MEMORY FILE NUMBER 2		MEMORY FILE NUMBER 3	
Register 600100	Data 1	**Register 600200**	Data 1	**Register 600300**	Data 1
Register 600101	Data 2	**Register 600201**	Data 2	**Register 600301**	Data 2
Register 600102	Data 3	**Register 600202**	Data 3	**Register 600302**	Data 3
Register 600103	Data 4	**Register 600203**	Data 4	**Register 600303**	Data 4

MEMORY FILE NUMBER 1		MEMORY FILE NUMBER 2		MEMORY FILE NUMBER 3	
Register 600104	Data 5	**Register 600204**	Data 5	**Register 600304**	Data 5

Para utilizar um Tag PLC para leitura do segundo registrador do *Memory File Number 2* do *Slave ID* 1 (um), utilize os seguintes parâmetros:

- **Configuração numérica**: Parâmetro *N1* igual a 1 (um), parâmetro *N2* igual a Oper, parâmetro *N3* igual a 2 (dois) e parâmetro *N4* igual a 201
- **Configuração por Strings**: Campo *Item* igual a 1:fr2.201

Para utilizar um Tag Bloco com 3 (três) Elementos para leitura dos 3 (três) últimos registradores do *Memory File Number 3* do *Slave ID* 2 (dois), utilize os seguintes parâmetros:

- **Configuração numérica**: Parâmetro *N1* igual a 2 (dois), parâmetro *N2* igual a Oper, parâmetro *N3* igual a 3 (três) e parâmetro *N4* igual a 302
- **Configuração por Strings**: Campo *Item* igual a 2:fr3.302

Para utilizar um Tag PLC para escrita no quarto registrador do *Memory File Number 1* do *Slave ID* 3 (três), utilize os seguintes parâmetros:

- **Configuração numérica**: Parâmetro *N1* igual a 3 (três), parâmetro *N2* igual a Oper, parâmetro *N3* igual a 1 (um) e parâmetro *N4* igual a 103
- **Configuração por Strings**: Campo *Item* igual a 3:fr1.103

NOTA

O protocolo Modbus permite efetuar leituras ou escritas de múltiplos sub-grupos (*file number*) de dados por meio de uma única requisição via funções **20** ou **21** respectivamente. No entanto, este Driver só permite manipular um sub-grupo de dados por vez em cada Tag PLC ou Bloco.

Partição Automática de Blocos

A partir da versão **2.0**, este Driver passa a contar com o recurso de **Partição Automática de Blocos**. Com este recurso, este 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 um equipamento, caso algum Tag Bloco exceda o tamanho máximo permitido.

A partir da versão **2.1**, este Driver passa também a suportar **Leitura por Superblocos**. Com este recurso habilitado, o usuário não precisa mais agrupar variáveis em Tags Bloco objetivando melhoria de desempenho, e é 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 é utilizado este Driver também não necessita usar o recurso de Partição Automática.

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

IMPORTANTE

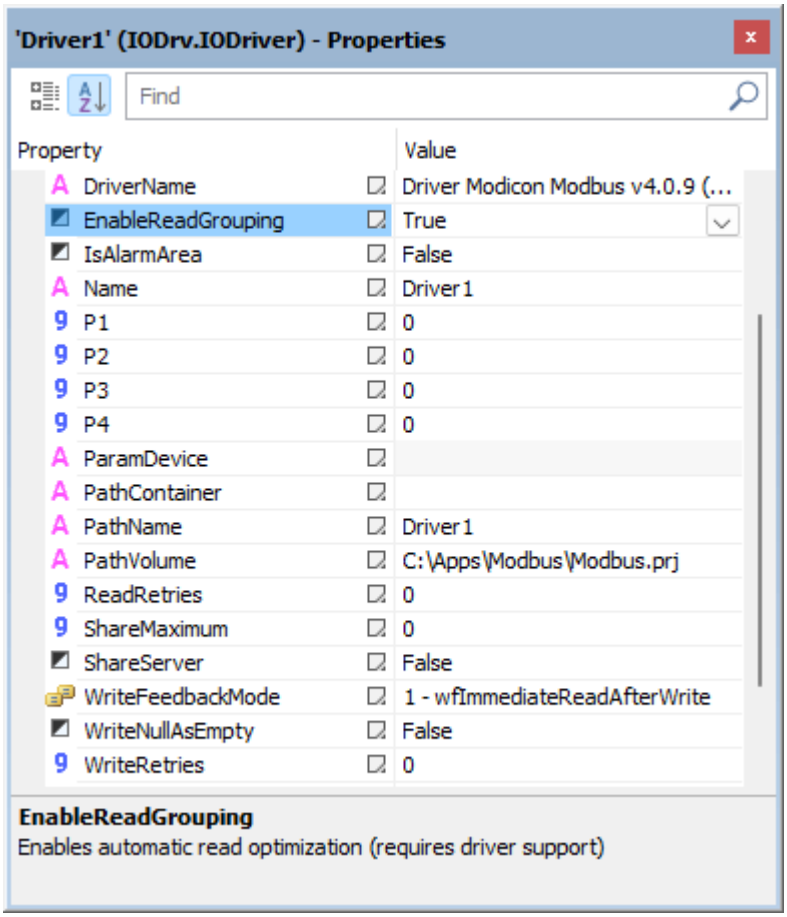
Tanto o agrupamento dos Superblocos no **Elipse E3**, no **Elipse Power** ou no **Elipse Water** como a Partição Automática de Blocos deste Driver requerem que o equipamento suporte os limites estabelecidos pelo Modbus padrão. Para mais informações, consulte o tópico **Limite Máximo para o Tamanho dos Blocos Suportado pelo Protocolo Modbus**. 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 **2.3** este Driver permite personalizar o limite máximo suportado para o PDU (*Protocol Data Unit*). Para isto, na janela de configurações deste 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. Para mais informações, consulte o tópico **Leitura por Superblocos (Agrupamento)**.

O artigo *KB-23112* do **Elipse Knowledgebase** apresenta um resumo das questões relativas ao agrupamento de Tags e dimensionamento de blocos neste Driver, discutidas neste e em outros tópicos. Para mais informações, consulte os tópicos **Leitura por Superblocos (Agrupamento)** e **Dicas de Otimização**.

Leitura por Superblocos (Agrupamento)

A partir da versão **2.1**, este Driver passa a suportar o recurso de **Leitura por Superblocos**. Este recurso é suportado pelo **Elipse E3**, pelo **Elipse Power** e pelo **Elipse Water**, podendo ser habilitado através da propriedade **EnableReadGrouping** de um 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, ou Tags PLC no **Elipse SCADA**, sem prejuízo de desempenho, pois a otimização do agrupamento na leitura é realizada automaticamente no momento da comunicação. A figura a seguir mostra a configuração da propriedade **EnableReadGrouping** no **Elipse E3**, no **Elipse Power** ou no **Elipse Water**.



Propriedade EnableReadGrouping

O **Elipse SCADA** não possui suporte a Superblocos. O comportamento da leitura de Tags no **Elipse SCADA** é idêntico ao do **Elipse E3**, do **Elipse Power** ou do **Elipse Water** quando a opção **EnableReadGrouping** está configurada em Falso. Em ambos os casos, este 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 considerar o agrupamento ao definir os Tags de uma aplicação, como descrito mais adiante neste tópico.

NOTA

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*.

IMPORTANTE

Tanto o agrupamento dos Superblocos no **Elipse E3**, no **Elipse Power** ou no **Elipse Water** como a **Partição Automática de Blocos** deste Driver requerem que um equipamento suporte os limites estabelecidos pelo Modbus padrão. Para mais informações, consulte o tópico **Limite Máximo para o Tamanho dos Blocos Suportado pelo Protocolo Modbus**. 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 **2.3** este Driver permite personalizar o limite máximo suportado para o PDU (*Protocol Data Unit*). Para isto, na janela de configurações deste Driver, aba **Modbus**, habilite a opção **Customize Max. PDU Size** e configure o tamanho máximo de bytes suportado para o PDU por um equipamento. Caso um equipamento suporte limites diferentes para cada tipo de função, é necessário realizar o agrupamento manual, conforme descrito 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, pode ser necessário efetuar alguns ajustes manuais para utilizar o recurso de Superblocos, caso seja viável. Para alguns casos específicos, o usuário tem a opção de montar estes Tags com um prefixo **DRG.** (*DisableReadGrouping*), para que estes Tags não sejam incluídos na leitura por Superblocos. As condições a seguir podem ser incompatíveis com a leitura por Superblocos:

- Equipamentos que definem limites máximos de tamanho de bloco inferiores ao limite padrão do protocolo, ou seja, **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. Nestes casos, caso seja necessário usar funções com limites diferentes, também é preciso desabilitar os Superblocos, ou seja, a propriedade **EnableReadGrouping** configurada para Falso, agrupando os Tags manualmente, conforme descrito mais adiante neste tópico.

- Equipamentos com descontinuidades, ou 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, agrupe estes Tags manualmente e use o prefixo **DRG.** (*DisableReadGrouping*) para que estes Tags não sejam incluídos na leitura por Superblocos

- Equipamentos que não suportam leitura em blocos. **Solução**: Desabilite os Superblocos, ou seja, configure a propriedade **EnableReadGrouping** para Falso, e defina Tags simples

- Equipamentos que só permitem a definição de blocos em endereços pré-determinados e com tamanhos fixos. **Solução**: Defina manualmente um Tag Bloco compatível com este tipo de endereço pré-determinado e tamanho fixo

determinado pelo manual de um equipamento e use o prefixo **DRG.** (*DisableReadGrouping*) para que este Tag não seja incluído na leitura por Superblocos

Agrupamento Manual

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

A partir da versão **4.0.18**, o usuário tem a opção de utilizar o prefixo **DRG.** (*DisableReadGrouping*) na configuração do campo **Item** (requer a **Configuração por Strings**) para selecionar quais Tags PLC ou Bloco não são incluídos no agrupamento de leitura por Superblocos por conta de alguma particularidade de um equipamento. Exemplos de configuração com o prefixo **DRG.** (*DisableReadGrouping*):

```
Tag.ParamItem = hr100 (agrupado por SuperBlocos)
Tag.ParamItem = DRG.hr100 (não é agrupado por SuperBlocos)
```

Ou quando a configuração incluir o *Slave ID*:

```
Tag.ParamItem = 1:hr100 (agrupado por SuperBlocos)
Tag.ParamItem = 1:DRG.hr100 (não é agrupado por SuperBlocos)
```

Na prática uma aplicação continua agrupando os Tags normalmente com leitura por Superblocos (propriedade **EnableReadGrouping** configurada para Verdadeiro), como exceção apenas dos Tags configurados com o prefixo **DRG.**, forçando uma transação individual sobre estes Tags.

O uso de **tipos de dados definidos pelo usuário**, também 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. Neste caso, 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 **Elipse Knowledgebase** apresenta um resumo das questões relativas ao agrupamento de Tags e dimensionamento de blocos neste Driver, discutidas neste e em outros tópicos.

Configurando Tags Especiais

Além dos **Tags de Comunicação**, este Driver suporta também alguns **Tags Especiais** que permitem a uma 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 deste Driver** na **configuração numérica**. Os tópicos **Forçar Wait Silence**, **Leitura do Código da Última Exceção** e **Leitura de Status do Modo Auto-Demotion** detalham os Tags Especiais suportados.

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 é detectado. Com este Tag, entretanto, é possível executar o serviço a qualquer momento por uma 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 0 (zero)
- **N2**: 9001
- **N3**: Não usado, pode ser deixado em 0 (zero)
- **N4**: Não usado, pode ser deixado em 0 (zero)
- **Valor**: Não usado, pode ser deixado em 0 (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 por este Driver em registradores internos, que podem ser acessados por meio deste Tag. Além disto, a cada comunicação bem sucedida com um determinado equipamento em que nenhuma exceção é retornada, este Driver zera automaticamente o registrador associado.

Códigos de Exceção

Os códigos de exceção são usados por um 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 de um **Mestre**, no caso deste 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 deste Driver, embora recebida corretamente, não pôde ser executada.

A especificação do protocolo Modbus define 9 (nove) códigos de exceção. A lista de exceções padrão do protocolo pode ser consultada no tópico **Lista de Exceções Padrão do Protocolo Modbus**. Além destes códigos, alguns fabricantes definem códigos adicionais, específicos de 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 identificador de um equipamento (*Slave Id*) seguido de dois pontos, como por exemplo "1:", "2:", "3:", e assim por diante
- **Item**: "LastExceptionCode"

Configuração Numérica

- **B1**: Endereço de um dispositivo **Escravo** (*Slave Id*)
- **B2**: 9999
- **B3**: Não usado, deve ser deixado em 0 (zero)
- **B4**: Não usado, deve ser deixado em 0 (zero)

Valores dos Elementos de Bloco retornados:

- **Elemento 1 (índice 0)**: Código da exceção retornada por um equipamento. Para mais informações, consulte o tópico **Lista de Exceções Padrão do Protocolo Modbus**
- **Elemento 2 (índice 1)**: Parâmetro *N2* ou *B2* do Tag de Comunicação que gerou a exceção
- **Elemento 3 (índice 2)**: Parâmetro *N3* ou *B3* do Tag de Comunicação que gerou a exceção
- **Elemento 4 (índice 3)**: Parâmetro *N4* ou *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 de um Tag Especial

A utilização mais comum deste Tag durante o *scan* normal dos Tags de funções é através de um evento **OnRead** de um 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 de um script, de forma a indicar que a exceção já foi tratada. O exemplo a seguir está 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
```

O exemplo a seguir está escrito em VBScript (**Elipse E3**, **Elipse Power** ou **Elipse Water**).

```
' 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 é 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. O exemplo a seguir está 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
```

O exemplo a seguir está escrito em VBScript (**Elipse E3**, **Elipse Power** ou **Elipse Water**).

```
' 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 0 (zero), também os parâmetros de um Tag cuja comunicação provocou a exceção. Caso estas informações não sejam necessárias, pode-se perfeitamente ler o mesmo registro através de um Tag simples, ou Tag PLC no **Elipse SCADA**, sem a necessidade de usar Tags Bloco. Neste caso, os procedimentos recomendados permanecem os mesmos.

Leitura de Status do Modo Auto-Demotion

Quando este Driver está configurado para comunicar com equipamentos **Escravos** em modo **Auto-Demotion**, utilize a leitura deste **Tag Especial** para monitorar o estado atual de qualquer um dos equipamentos **Escravos** da rede Modbus, conforme os parâmetros a seguir.

Configuração por Strings

- **Dispositivo**: Valor numérico do identificador de um equipamento (*Slave Id*) seguido de dois pontos, como por exemplo "1:", "2:", "3:", e assim por diante
- **Item**: "DemotionStatus"

Ou:

- **Dispositivo**: "" (**String** vazia)
- **Item**: "n:DemotionStatus", em que *n* representa o identificador de um equipamento (*Slave Id*) seguido de dois pontos

Configuração Numérica

- **N1**: Endereço de um equipamento **Escravo** (*Slave Id*)
- **N2**: 9500
- **N3**: Não usado, deixe em 0 (zero)
- **N4**: Não usado, deixe em 0 (zero)

Os valores possíveis de retorno para indicar o atual estado do modo **Auto-Demotion** de um equipamento **Escravo** são **0**: Estado normal de comunicação, ou seja, o equipamento não está em modo **Demotion** ou **1**: Estado inativo de comunicação, ou seja, o equipamento está em modo **Demotion** e temporariamente fora de comunicação após exceder o limite de erros por *time-out*.

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

Funções Especiais para Leitura de Memória de Massa

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

- **GE SOE**: Realiza a coleta de eventos de equipamentos 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 de um equipamento (*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 de Controladores GE PAC RX7

O *buffer* de eventos pode ser lido através de **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 deste 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 de controladores GE PAC RX7 usando a configuração numérica, deve ser definida, na janela de configuração deste 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 em um equipamento

A cada *scan* neste Tag, este Driver verifica se existem eventos no *buffer* de um controlador. Se há eventos, este 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* por este Driver, este Tag reportado por eventos retorna o conjunto de eventos lidos nesta varredura.

Os eventos retornados geram uma sucessão de eventos **OnRead** neste Tag. Para cada evento lido, o **Elipse E3**, o **Elipse Power** ou o **Elipse Water** atualiza os campos de um Tag, os valores de Elementos e estampas de tempo, com os valores de um determinado evento, e executa uma vez um evento **OnRead**. O script do evento executado deve ser definido pelo usuário, e é geralmente usado para inserir os dados de um Tag em um Histórico.

Cada evento é representado por um Bloco de 2 (dois) Elementos, com o campo **Timestamp** lido de um equipamento. Os campos do respectivo Tag Bloco de leitura são mostrados na tabela a seguir.

Campos de um Tag Bloco

OFFSET	SIGNIFICADO	TIPO DE DADOS	FAIXA DE VALORES
0	Identificação do ponto	Byte	Entre 0 (zero) e 15
1	Status do ponto	Byte	Entre 0 (zero) e 1 (um)

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

IMPORTANTE

Ao ler eventos de memória de massa em Tags reportados a eventos no **Elipse E3**, no **Elipse Power** ou no **Elipse Water**, desabilite a banda morta de um Tag, ou seja, configure a propriedade **EnableDeadBand** para Falso, e também do objeto Histórico associado, ou seja, configure a propriedade **DeadBand** para 0 (zero), para evitar a perda de eventos com valores próximos. Também é importante desabilitar o histórico por varredura, ou seja, no **Elipse E3**, no **Elipse Power** ou no **Elipse Water** configure a propriedade **ScanTime** para 0 (zero). Com isto, garante-se que novos eventos só são armazenados através do método **WriteRecord**, executado no evento **OnRead** de um Tag, evitando a duplicação de eventos.

Tags Reportados por Eventos por Ponto

A partir da versão **2.5** deste 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 Tag anterior, o valor retornado possui apenas 1 (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 de um 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 de um ponto, como por exemplo configure o parâmetro *N3* com o valor 202 para o ponto 2 (dois)
- **N4**: Endereço base da pilha de eventos em um equipamento

Para mais informações sobre Tags reportados por evento, consulte o tópico específico no *Manual do Usuário do Elipse 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 deste Driver a cada leitura de eventos de um equipamento, com as respectivas estampas de tempo lidas deste equipamento. Este Tag utiliza os seguintes parâmetros:

Configuração por Strings

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

Configuração Numérica

- **N1**: *Slave ID*
- **N2**: Código de uma operação
- **N3**: 100 + Índice de um ponto
- **N4**: Endereço base da pilha de eventos em um equipamento
- **Valor**: Status de um ponto

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

Para a leitura de relés SEPAM, o modelo de *offset* de endereçamento deve ser configurado selecionando a opção **Data is addressed from 0** na aba **Modbus**. A leitura de eventos é realizada com a utilização dos seguintes 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 de um equipamento. Este Tag, além de retornar todos os eventos lidos à aplicação reportados a evento, ainda acumula os eventos lidos no *buffer* interno deste 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 de um equipamento, mas sim retorna os eventos do *buffer* interno deste 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 a varredura ou *scan* habilitada. Este Tag é útil quando o usuário precisa obter eventos de um tipo e de uma fonte específicas, tais como relé, zona, endereço e tipo. Um exemplo de utilização é uma 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, usá-lo poupa o usuário de 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** em uma aplicação

Uma 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 de um equipamento. A seguir é descrita a configuração destes Tags.

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

Este Tag é reportado a eventos. Uma aplicação típica é a inserção de eventos em um objeto Histórico associado, através do método **WriteRecord** deste Histórico, executado no evento **OnRead** de um Tag. A cada leitura, ou seja, a cada período de varredura de um Tag, este Driver pode coletar até 4 (quatro) novos eventos de um equipamento. Este é o número máximo de eventos que cada zona de eventos de um relé dispõe a cada requisição de leitura.

Como é durante a leitura deste Tag que os eventos são de fato coletados de um equipamento, mesmo que os dados não sejam utilizados diretamente, ou seja, mesmo **que não seja preciso armazenar todos os eventos coletados em Histórico, a 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 3 (três) Elementos, da seguinte forma:

Configuração por Strings

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

Por exemplo, para a leitura da Zona 1 (um) do Escravo 1 (um), **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". Para mais informações, consulte o tópico **Configuração por Strings**.

Configuração Numérica

Para usar a configuração numérica, defina 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 de um escravo (relé) na rede, entre 1 (um) 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 Elementos de Bloco, que têm os valores retornados como reportados a eventos.

Significados dos Elementos de Bloco (tipo de dados SP_events)

OFFSET	SIGNIFICADO	TIPO DE DADOS	FAIXA DE VALORES
0	Tipo de evento	**Word**	De 0 (zero) a 65535 (800H para **Remote Annunciation**, **Internal Data** e **Logic Input**)
1	Endereço de um evento	**Word**	Referencia endereços de bits de 1000H a 105FH
2	Borda de subida ou de descida	**Word**	**00**: Borda de descida ou **01**: Borda de subida

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

IMPORTANTE

Ao ler eventos de memória de massa em Tags reportados a eventos no **Elipse E3**, no **Elipse Power** ou no **Elipse Water**, desabilite a banda morta de um Tag, ou seja, configure a propriedade **EnableDeadBand** para Falso, e também no objeto Histórico associado, ou seja, configure a propriedade **DeadBand** em 0 (zero) para evitar a perda de eventos com valores próximos. Também é importante desabilitar o histórico por varredura, ou seja, no **Elipse E3**, no **Elipse Power** ou no **Elipse Water** configure a propriedade **ScanTime** em 0 (zero). Com isto, garante-se que novos eventos só são armazenados através do método **WriteRecord**, executado no evento **OnRead** de um Tag, evitando a duplicação de eventos.

Tag de Leitura de Evento Único

Este Tag também é reportado a eventos, e é possível usar o evento **OnRead** para armazenamento em Histórico. Note que nada impede, entretanto, que seja tratado como um Tag normal, ou um Tag de tempo real, caso só interesse o 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 lê eventos de um equipamento, mas sim de um *buffer* interno deste Driver, previamente preenchido durante a leitura de um Tag de coleta de todos os eventos. Este Tag retorna apenas um Elemento, reportado a eventos, podendo ser configurado como Tag simples e não um um Tag Bloco. Este Driver aceita no máximo 8 (oito) eventos acumulados por ponto de evento, ou seja, para cada combinação de relé, zona, tipo e endereço do evento no *buffer* interno. Se ocorrer *overflow*, ou seja, se mais de 8 (oito) eventos de um mesmo ponto são retornados sem que nenhum Tag de evento único os colete, este 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 ou *scan* dos Tags de evento único com um valor equivalente à metade do configurado para um Tag de coleta de todos os eventos associados, evitando-se assim a perda de eventos por *overflow* do *buffer* interno deste Driver.

Este Tag deve ser configurado com os seguintes parâmetros:

Configuração por Strings

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

Por exemplo, para a leitura de eventos do tipo **800H** no endereço 1 (um) da zona 2 (dois), **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". Para mais informações, consulte o tópico **Configuração por Strings**.

Configuração Numérica

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

Opções para o endereço de evento (Events Zone Offset)

ZONA DE EVENTOS	EVENTS ZONE OFFSET
1	0 (zero)

ZONA DE EVENTOS	EVENTS ZONE OFFSET
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**, no **Elipse Power** e no **Elipse Water**, deve-se usar o prefixo "&H", como por exemplo &H10 = 16. No **Elipse SCADA**, use o sufixo "h", como 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 um evento foi de fato lido de um relé, durante a leitura do Tag de leitura de todos os eventos descrito anteriormente

Para mais informações sobre os eventos de um relé, significados e endereçamento, consulte a documentação do fabricante. Para mais informações sobre Tags reportados a eventos, consulte o tópico **Tags Reportados a Eventos** no *Manual do Usuário do Elipse E3*.

Algoritmo de Leitura de SOE Genérico da Elipse Software

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

O algoritmo de Sequenciamento de Eventos (SOE ou *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 neste Driver.

A leitura de eventos neste Driver segue o procedimento padrão, definido pela **Elipse Software**, lendo eventos de **tabelas montadas na memória de um equipamento** ou de dispositivos **Escravos** pelo software residente ou aplicação (*ladder*).

Para utilizar este algoritmo é preciso definir Tags com a **função especial Gen SOE** deste Driver, o que pode ser realizado tanto através da antiga **configuração numérica**, ou parâmetros *N* e *B*, como pelo novo método de

configuração por Strings, nos campos **Dispositivo** e **Item**. A configuração dos Tags é descrita mais adiante no tópico **Procedimento de Aquisição em uma 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 de um 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**, pois o contrário é mais comum, através do campo **Write** da operação, na aba **Operations**.

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

Um 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 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 por este Driver a cada procedimento de coleta ou *download* de eventos.

O usuário pode definir formatos distintos de dados, ou 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** deste Driver. Neste caso, não é possível definir um campo **Timestamp** em um equipamento, pois a estampa de tempo vem com a data da leitura, e o evento tem somente um campo, podendo ser representado por um Tag simples ou um 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 de um 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 **Tabela de Eventos**, **Procedimento de Aquisição em um Equipamento** e **Procedimento de Aquisição em uma Aplicação** descrevem de forma detalhada este algoritmo, a implementação no software de um equipamento (*ladder*) e como realizar a leitura usando os Tags deste Driver, respectivamente.

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 endereço inicial, ou endereço base, contíguo aos registradores de controle, e pelo número máximo de registros, que define o limite final. A tabela a seguir ilustra a disposição dos eventos dentro de um *buffer* circular de uma tabela.

Disposição de eventos de um buffer circular

EVENTO	TIMESTAMP	ELEMENTO 1	ELEMENTO 2	ELEMENTO 3	...	ELEMENTO N
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 a estampa de tempo. Este campo, cuja presença não é obrigatória e que não precisa necessariamente aparecer na primeira posição, define a propriedade **Timestamp** de um Tag, e não é retornado nos Elementos. Para mais informações, consulte o tópico **Tipos de Dados Definidos pelo Usuário**.

É possível também definir eventos com **tipos de dados nativos deste Driver**, e neste caso tem-se apenas 1 (um) Elemento de dado em cada evento, sem estampa de tempo.

Os eventos devem ser inseridos em um *buffer* circular em ordem crescente, retornando ao endereço base após atingir o limite superior deste *buffer* circular. Os registros de controle a seguir devem ser definidos para cada tabela:

- **Status da tabela**: Deve ser mantido pelo equipamento, indicando o número de eventos disponíveis para a leitura neste *buffer* circular. Deve ser atualizado pelo equipamento sempre que novos eventos são adicionados a este *buffer* circular, ou após a conclusão da coleta de eventos por uma 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 um equipamento deve inserir o próximo evento. Deve ser incrementado pelo equipamento a cada nova inserção de eventos neste *buffer* circular, voltando ao endereço base após alcançar o limite superior deste *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 neste *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 por este Driver a cada leitura individual de eventos. Após cada leitura, este Driver escreve neste registro o número de registros que conseguiu ler. A aplicação residente em um **Escravo**, ou *ladder*, deve então imediatamente subtrair o valor escrito por este 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 deste Driver e deve ser deixado em 0 (zero) na versão atual

Conforme já mencionado, o endereço base de um *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 a 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 endereços mostrados na tabela a seguir 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 em um Equipamento** é descrito passo a passo o procedimento ou algoritmo de aquisição sob o ponto de vista de um equipamento **Escravo** (CLP). No tópico seguinte, **Procedimento de Aquisição em uma Aplicação**, discute-se como configurar uma aplicação para a aquisição dos eventos desta tabela.

Procedimento de Aquisição em um Equipamento

Neste tópico discute-se o algoritmo de coleta de eventos sob o ponto de vista de um equipamento ou dispositivo **Escravo**. O objetivo é deixar claro ao desenvolvedor o que é preciso ser implementado no software residente de um equipamento, ou *ladder*.

Um equipamento 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*.

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

Se o número de eventos disponíveis é maior que o **máximo permitido em um único frame de comunicação** do protocolo, este 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 um 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 por este Driver no **Status da Aquisição**, a aplicação de um equipamento 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, este Driver pode iniciar uma nova aquisição a qualquer momento.

Um equipamento pode inserir novos eventos na tabela durante o processo de aquisição por este equipamento, 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** é zerado. Os eventos coletados são então disponibilizados à aplicação por meio de Tags reportados a eventos, 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 de um equipamento. 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.

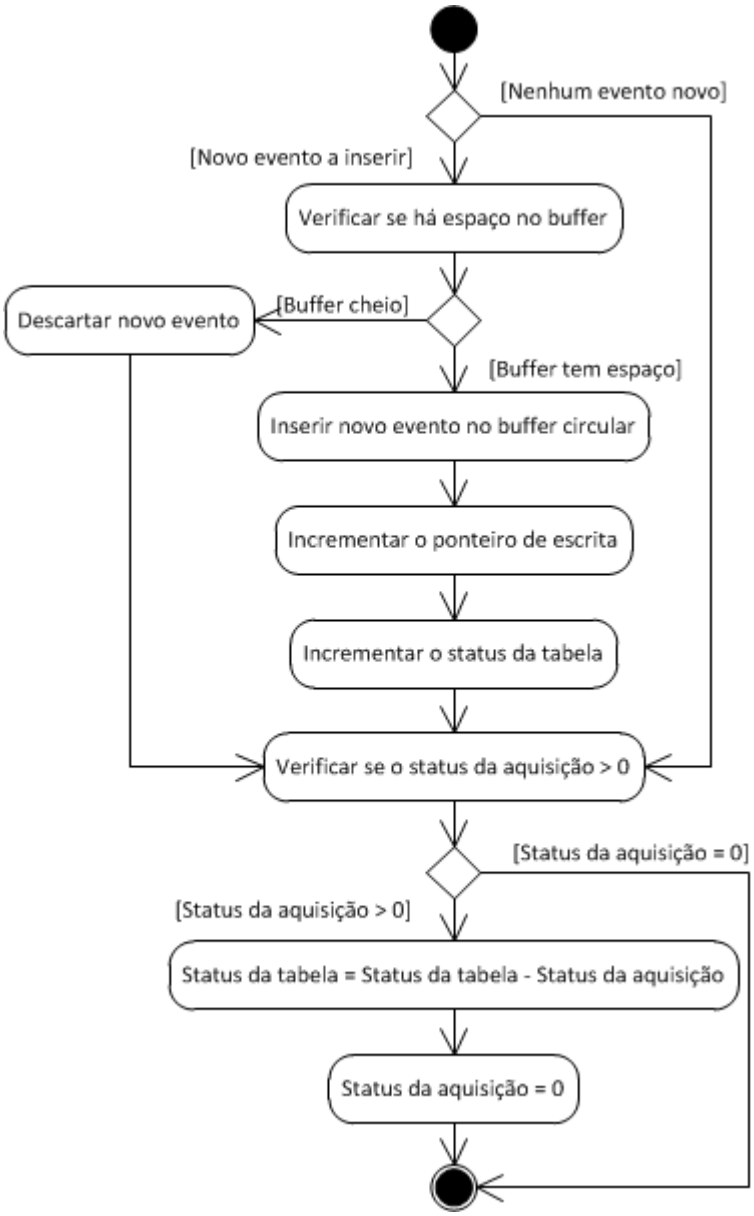


Diagrama de Atividade UML (Equipamento)

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**.

Ao usar estruturas ou tipos de dados definidos pelo usuário, é possível associar a cada evento uma estampa de tempo, ou *timestamp*, fornecida por um equipamento. Neste caso, o valor do campo **Timestamp** deve ser fornecido em um campo da estrutura, na memória de um equipamento, na ordem em que é declarado no arquivo de configuração, e o valor não é mostrado em nenhum Elemento de Bloco, apenas retornado 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 por este Driver pode ser usado. O tipo de dados **GenTime**, entretanto, foi criado especialmente para uso com o **Elipse Modbus SOE**, devido à facilidade de definição no software residente, ou *ladder*, de um equipamento.

Caso não seja necessária uma precisão de milissegundos, outra opção a considerar é o tipo de dados **UTC32** deste Driver, representado como um inteiro de apenas 32 bits, ou 4 (quatro) bytes, com os segundos desde 1/1/1970, sem a representação dos milissegundos, considerados como 0 (zero).

O tópico a seguir, **Procedimento de Aquisição em uma Aplicação**, descreve como configurar uma aplicação para a coleta dos eventos acumulados em um equipamento ou dispositivo **Escravo** programável.

Procedimento de Aquisição em uma Aplicação

Neste tópico é detalhada a configuração de uma aplicação para a aquisição dos eventos acumulados em um equipamento ou dispositivo **Escravo** programável.

A leitura dos eventos em uma aplicação é realizada por meio de Tags que usam a função especial de leitura **Gen SOE**. O tipo de dados de um Tag define a estrutura dos eventos armazenados na tabela de um equipamento. Se um **tipo de dados nativo deste Driver**, ou *built-in type*, é definido, cada evento tem apenas um elemento deste tipo de dados, sem estampa de tempo fornecida por um equipamento, pois a estampa de tempo representa o momento da coleta dos eventos. Por outro lado, se são usados **tipos de dados definidos pelo usuário**, é possível definir estruturas para os eventos, incluindo estampas de tempo, como visto mais adiante neste tópico.

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

Configuração por Strings

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

Em que:

- **N**: Tamanho da tabela em um equipamento, 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. Para mais informações, consulte o tópico **Configuração por Strings**
- **tam. do tipo**: Usado apenas para tipos de dados de tamanho variável. Para mais informações, consulte o tópico **Configuração por Strings**
- **byte order**: Ordenamento de bytes. Deve ser omitido para equipamentos que seguem plenamente o padrão do protocolo. Para mais informações, consulte o item **Byte Order** no tópico **Configuração por Strings**. Quando são usadas estruturas, afeta apenas os elementos individuais. Para mais informações, consulte o tópico **Tipos de Dados Definidos pelo Usuário**
- **bit**: Mascaramento de bits. Em geral pode ser omitido, pois dificilmente é usado aqui. Para mais informações, consulte o campo **Bit** no tópico **Configuração por Strings**

Exemplo:

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

O tipo de dados **TYPE3** está definido no formato a seguir no arquivo padrão de exemplo deste Driver. Para mais informações, consulte 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 6 (seis) campos de dados e estampa de tempo. Portanto, deduz-se que este Tag deve ser um Tag Bloco com 6 (seis) Elementos para representar esta estrutura.

Note que, conforme já explicado, o valor da estampa de tempo, embora ocupe registros em um equipamento, não necessita de Elementos de Tag Bloco, pois este valor é retornado na propriedade **Timestamp** deste Tag.

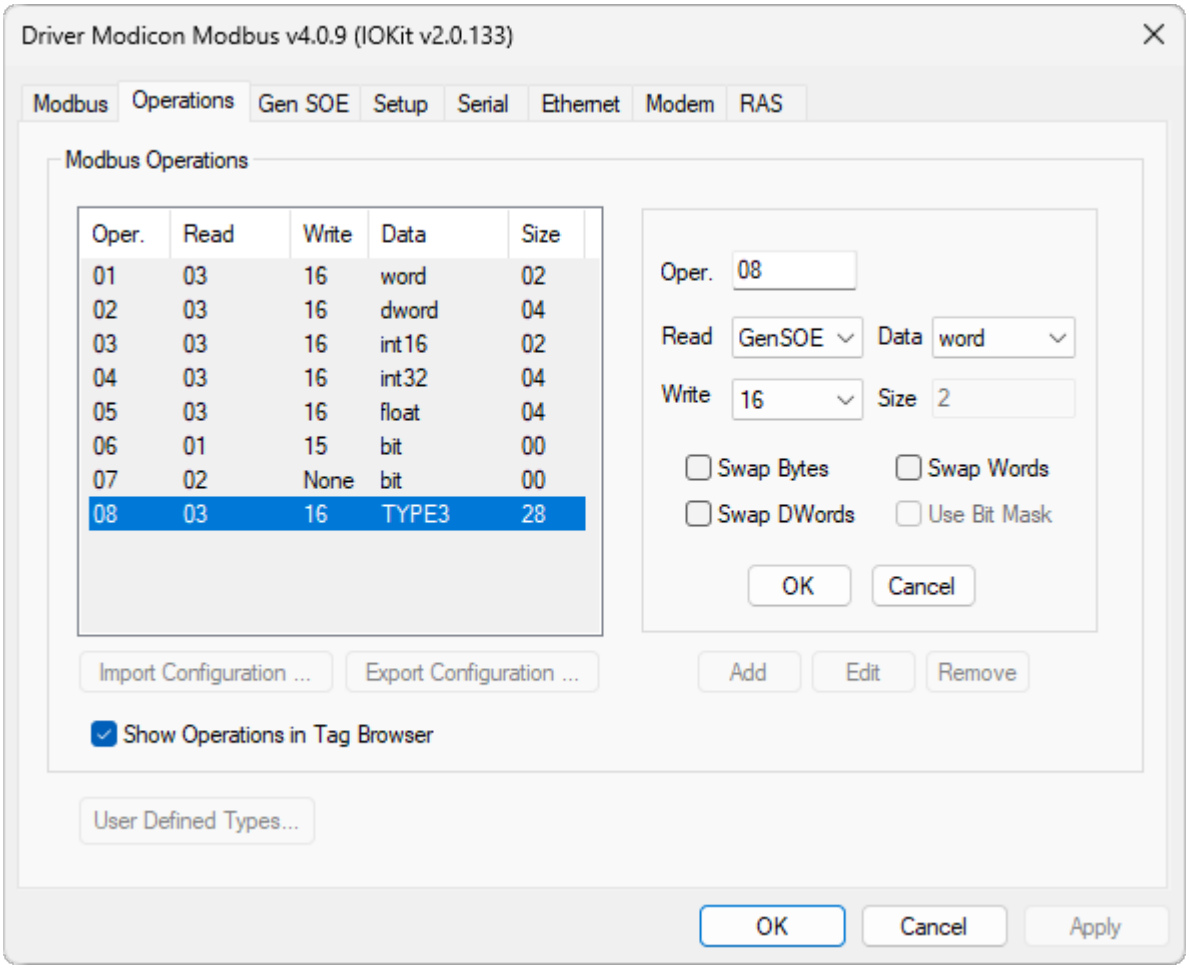
NOTA

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 **Elipse SOE** usando a configuração numérica, é necessário configurar uma operação na aba **Operations** usando a **função especial GenSOE**.

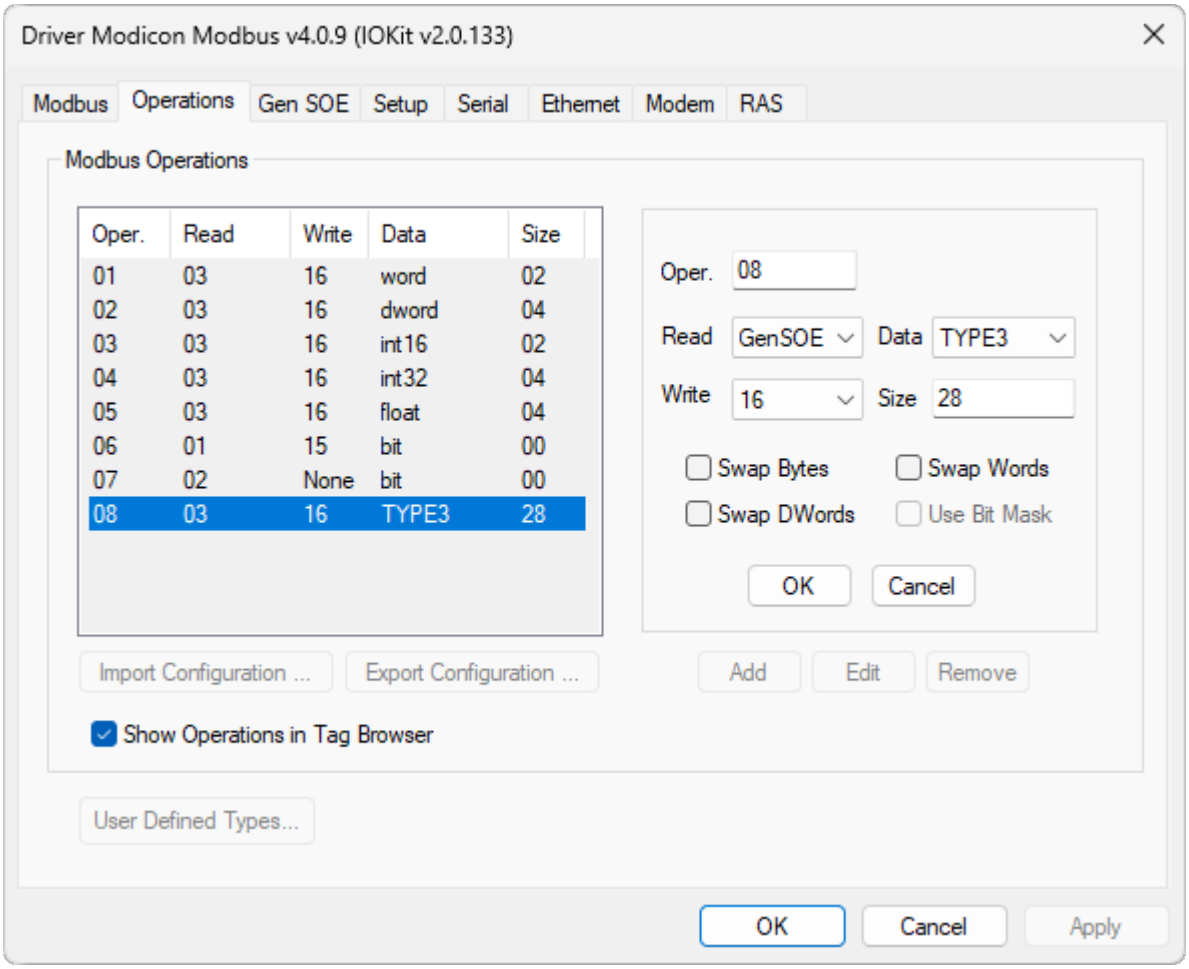
A figura a seguir mostra um exemplo de adição de operação utilizando a função especial **GenSOE** com o tipo de dados **Word**.



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 seja suportada por um equipamento.

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



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

O tipo de dados **TYPE3** está definido no formato a seguir no arquivo de exemplo deste 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 6 (seis) campos de dados e estampa de tempo, e com endereço padrão, o parâmetro *B4* deste Tag, igual a "101H", ou 257 em decimal. Para a leitura, portanto, é preciso definir um Tag Bloco de 6 (seis) Elementos com a seguinte configuração:

- **B1**: Endereço de um equipamento **Escravo** (CLP) na rede (*Slave Id*)
- **B2**: 8 (oito, operação definida anteriormente com a função especial **GenSOE**)
- **B3**: N (tamanho da tabela em um equipamento, 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 (seis)

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 **Elipse E3**, do **Elipse Power** ou do **Elipse Water** seja usado para inserir um Tag em uma aplicação, como explicado no tópico **Tipos de Dados Definidos pelo Usuário**, os Elementos deste Tag já são nomeados conforme o nome dado aos elementos da estrutura na declaração. O Tag Browser pode ser aberto clicando-se em na aba **Design** deste Driver.

Utilização

Uma vez tendo definido um Tag, ou Tags, apropriado, habilite a varredura e deixe para este Driver a tarefa de coletar os eventos da respectiva tabela, sempre que novos eventos sejam detectados.

Tags associados à função **GenSOE**, ou **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 a este Driver retornar vários eventos em uma única operação de leitura, ou seja, em um único intervalo da varredura de um Tag.

Isto significa que este Driver retorna o conjunto de eventos, ou no caso do exemplo anterior conjuntos de blocos com 6 (seis) campos de dados e estampa de tempo, de uma vez só, o que produz uma sequência de eventos **OnRead** em um Tag, um para cada evento, ou bloco com 6 (seis) campos de dados e estampa de tempo, retornado por este 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 Elipse 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 no evento **OnRead** o método **WriteRecord** do objeto Histórico previamente associado, garantindo assim a gravação de todos os eventos que cheguem a este Histórico. Neste caso, este Histórico deve ser configurado sem banda morta, ou seja, configurar a propriedade **DeadBand** com o valor 0 (zero), e desabilitar o histórico por *scan*, ou seja, no **Elipse E3**, no **Elipse Power** ou no **Elipse Water** configurar a propriedade **ScanTime** com o valor 0 (zero). A propriedade **EnableDeadBand** deste Tag também deve ser configurada para Falso.

IMPORTANTE

Ao ler eventos de memória de massa em Tags reportados a eventos no **Elipse E3**, no **Elipse Power** ou no **Elipse Water**, desabilite a banda morta do Tag, ou seja, configure a propriedade **EnableDeadBand** com o valor Falso, e também no objeto Histórico associado, ou seja, configure a propriedade **DeadBand** com o valor 0 (zero) para evitar a perda de eventos com valores próximos. Também é importante desabilitar o histórico por varredura, ou seja, no **Elipse E3**, no **Elipse Power** ou no **Elipse Water** configure a propriedade **ScanTime** com o valor 0 (zero). Com isto, garante-se que novos eventos só são armazenados através do método **WriteRecord**, executado no evento **OnRead** de um Tag, evitando a duplicação de eventos.

Otimização e Compatibilidade

Alguns equipamentos, como os equipamentos da marca ATOS, não suportam a leitura em blocos de tipos de dados de estruturas diferentes. Na prática, isto impede que este Driver leia em um bloco único dados de registradores de controle e de eventos. Para coletar eventos de equipamentos 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 de um medidor ABB.

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

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

Este Driver também conta com 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 deste equipamento.

Configuração por Strings

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

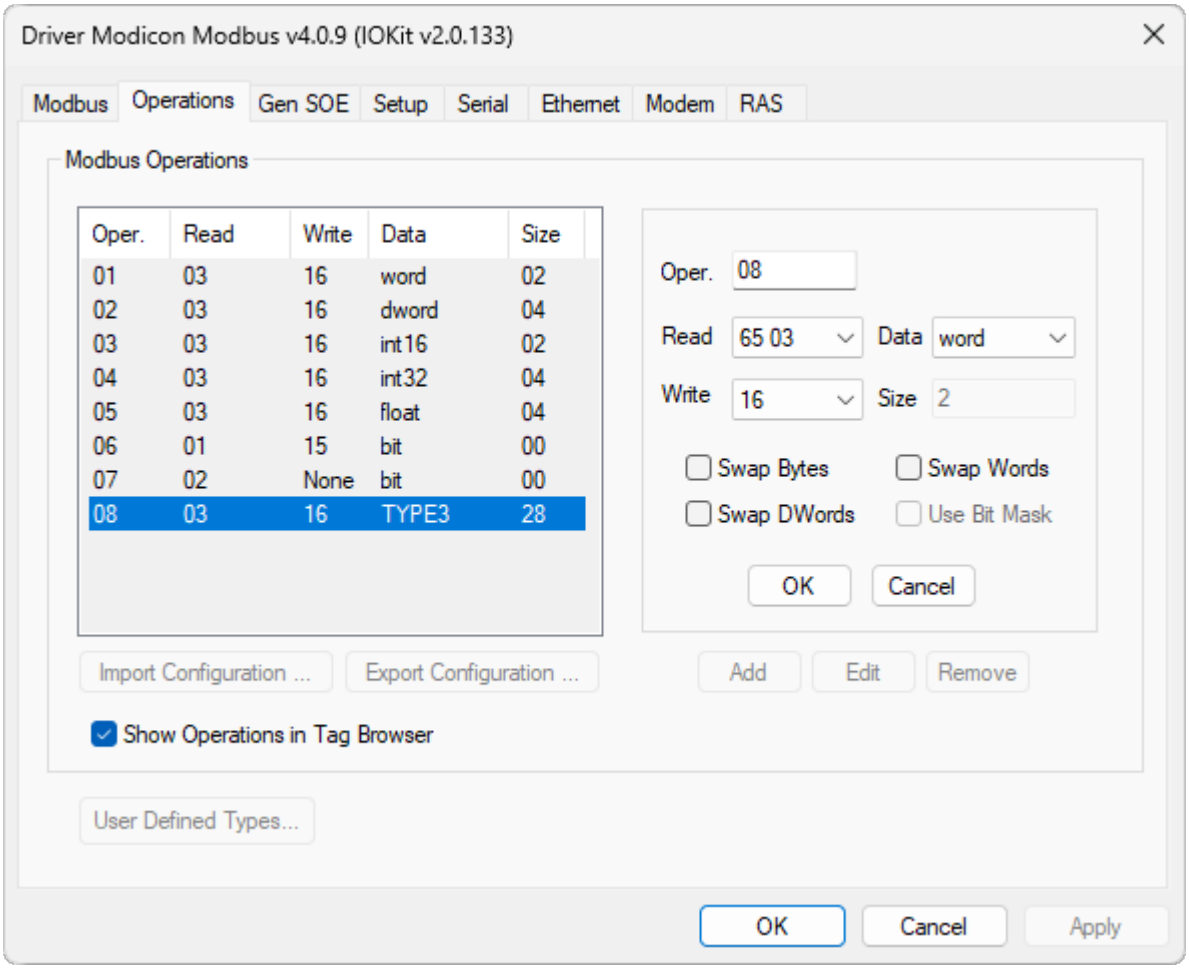
Em que:

- **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, portanto prefira utilizar as máscaras de bit de uma aplicação. Para mais informações, consulte o tópico **Configuração por Strings**

Configuração Numérica (Parâmetros N ou B)

- **N1 ou B1**: Slave Id
- **N2 ou B2**: Código da **operação** configurada com a função **65 03**. Consulte mais informações a seguir
- **N3 ou B3**: Não usado, deixe em 0 (zero)
- **N4 ou B4**: Endereço de um registrador

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



Criação de uma operação com a função especial 65 03

Apêndice

Esta seção contém informações adicionais sobre o protocolo Modbus.

Dicas de Otimização

Este tópico enumera algumas dicas de otimização para a comunicação com equipamentos **Escravos**.

Dicas de Configuração deste Driver para o Elipse E3, para o Elipse Power e para o Elipse Water

- Use Superblocos sempre que possível, dando preferência à criação de Tags simples, ou Tags PLC no **Elipse SCADA**, ao invés de Tags Bloco. Para mais informações, consulte o tópico **Leitura por Superblocos (Agrupamento)**.
- Se não é possível usar Superblocos, dê preferência à criação de Tags Bloco, agrupando o maior número possível de variáveis no menor número de blocos. Para mais informações, consulte o texto sobre o agrupamento manual no tópico **Leitura por Superblocos (Agrupamento)**.
- Considere as recomendações do artigo *Dicas de Performance para o Elipse E3* no **Elipse Knowledgebase**.
- No caso de redes com alta latência, banda reduzida ou perda de pacotes, consulte também o artigo *Configurações de rede do Elipse E3 para redes com alta latência, banda reduzida e/ou perda de pacotes*.
- Em redes de alta latência, configure *time-outs* mais elevados considerando 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. Caso as falhas por *time-out* de um determinado equipamento **Escravo** sejam muito frequentes, considere habilitar o modo **Auto-Demotion** para que este equipamento permaneça temporariamente fora da comunicação.

Dicas de Configuração deste Driver para o Elipse SCADA

- Dê preferência à criação de Tags Bloco, agrupando o maior número possível de variáveis no menor número de blocos. Para mais informações, consulte o texto sobre o agrupamento manual no tópico **Leitura por Superblocos (Agrupamento)**.
- Considere as recomendações do artigo *Desenvolvendo aplicativos Elipse SCADA visando o melhor desempenho possível*, também disponível no **Elipse Knowledgebase**.
- Em redes de alta latência, configure *time-outs* mais elevados considerando 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. Caso as falhas por *time-out* de um determinado equipamento **Escravo** sejam muito frequentes, considere habilitar o modo **Auto-Demotion** para que este equipamento permaneça temporariamente fora da comunicação.

Dicas para a Configuração ou Programação de um Equipamento

- Se possível, agrupe as variáveis definidas por uma aplicação residente, ou *ladder*, que possuam menor tempo de varredura, ou *scan*, em endereços contíguos na memória de um equipamento. O tempo total de varredura dos Tags depende muito da capacidade de agrupamento das variáveis em blocos de comunicação.

Dúvidas Mais Frequentes

Particularidades conhecidas de um equipamento Twido da empresa Schneider

- Uma aplicação está tentando ler um valor do tipo de dados **Float**, porém não está conseguindo. O valor mostrado em um equipamento está totalmente diferente do mostrado nesta aplicação para o mesmo endereço.
  - **Resposta**: Este equipamento não utiliza o ordenamento de bytes padrão do protocolo, ou *big endian*. Deve-se configurar o ordenamento de bytes executando o *swap* ou 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**. Para mais informações, consulte o tópico **Aba Operations**.
- Uma aplicação está tentando ler as entradas e saídas de um equipamento, porém não está conseguindo.
  - **Resposta**: Este equipamento não permite leitura ou escrita nas variáveis de entrada e saída, e é necessário utilizar variáveis internas a este equipamento para realizar esta leitura, isto é, cria-se um espelho das entradas e saídas em uma área onde este Driver tenha acesso. Deve-se ainda ter o cuidado de criar uma rotina neste equipamento para verificar quando o valor de uma saída é alterado por uma aplicação para que seja realmente ativada ou desativada neste equipamento.

Particularidades conhecidas de um equipamento MPC 6006 da empresa Atos - Schneider

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

Como juntar dois valores do tipo de dados Int16 em um equipamento em um valor do tipo de dados Int32 em uma aplicação?

- Existe um número de 32 bits que está armazenado em dois registradores de 16 bits cada um em um equipamento. Como mostrar na tela de uma 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 deste Tag de Comunicação, deve-se informar o primeiro endereço de cada variável em um equipamento. Para mais informações, consulte o tópico **Configurando um Tag de Comunicação**. Com isto, este Driver une dois registros de 16 bits de um equipamento em um único valor de 32 bits, retornado na propriedade **Value** de um Tag ou em um Elemento de um Tag Bloco. Caso esteja usando a configuração por **Strings**, os campos **Dispositivo** e **Item**, informe o tipo de dados logo após o endereço do registro. Para mais informações, consulte o tópico **Configuração por Strings**. Caso esteja utilizando a antiga **configuração numérica**, os parâmetros *N* ou *B*, é necessário definir operações com tipos de dados de 32 bits. Note que, na aba **Operations** da janela de configuração, os tipos de dados de 32 bits são sempre mostrados com tamanho, o campo **Size**, de 4 (quatro) bytes. Para mais informações, consulte o tópico **Tipos de Dados Suportados**.

- Uma aplicação já está desenvolvida, porém é preciso juntar os valores de 2 (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, como por exemplo **UInt32 = (HighWord × 65536) + LowWord**.

- Uma aplicação precisa ler valores do tipo de dados **Float**. A função de leitura foi configurada como **03** e escrita **16** com o tipo de dados **Float**. Porém, uma aplicação **Elipse E3**, **Elipse Power** ou **Elipse Water** mostra um valor que não condiz com o valor que está em um equipamento.

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

Como 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 comunicar com cada um destes equipamentos?

  - **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 RS-485, todos os equipamentos escutam ao mesmo tempo as requisições deste Driver, ou seja, há um barramento único, porém apenas o que possuir o *Slave Id* correspondente responde à requisição, pois não pode haver múltiplos equipamentos com o mesmo identificador. Na configuração por **Strings**, este valor pode ser fornecido no campo **Dispositivo**, ou no início do campo **Item**. Para mais informações, consulte o 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 mais informações sobre a instalação e manutenção de redes seriais é o livro *Serial Port Complete*, de Jan Axelson.

- Existe mais de uma porta serial em um computador. Como configurar este 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, como por exemplo Serial 1, Serial 2 e assim por diante, são necessários tantos Drivers de Comunicação quantas portas existirem. As configurações dos **Tags** deste Driver podem ser as mesmas para todos os objetos ou instâncias deste 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 de porta a ser usada são realizadas na aba **Serial** da janela de configurações deste Driver. Para mais informações, consulte o tópico **Propriedades**.

Como comunicar com mais de um equipamento em uma rede de comunicação Serial com conversor para RS-485?

- Uma rede RS-485 tem vários equipamentos comunicando através de um conversor RS232-RS485 pela porta Serial. Sempre que ocorre uma troca de endereço (*Slave ID*), ou seja, quando este Driver vai solicitar dados de outro equipamento, ocorre um *time-out*. Após retentar a mesma mensagem, um equipamento responde normalmente. Existe alguma forma de evitar este *time-out* durante a troca de endereço ou *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** da biblioteca **IOKit**, disponível na **janela de configurações**. Esta opção 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** da biblioteca **IOKit** pode trazer significativo prejuízo de performance em algumas aplicações, e deve ser utilizada apenas quando absolutamente necessário. Certifique-se de que o conversor está em boas condições e se de fato exige a utilização deste *delay*. Se necessário, contate o suporte do fabricante.

Como 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 comunicar com cada um destes equipamentos?

  - **Resposta**: Atualmente, para cada endereço IP, é necessário tantos Drivers de Comunicação quantos endereços IP deseja-se comunicar. A configuração referente aos **Tags** deste 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 / RS-485 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** da biblioteca **IOKit**, na **janela de configurações** deste 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**, as camadas física, de rede e de transporte, como também convertem o **Modbus RTU** em **Modbus TCP**, a camada de aplicação. Em último caso, se é 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 este Driver?

  - **Resposta**: Sim, existem diversas alternativas. A **Elipse Software** disponibiliza uma versão gratuita, ou de demonstração, 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 *http://www.plcsimulator.org*. Existem ainda outras opções e uma lista de outras aplicações pode ser encontrada no site da *Modbus.org*.

Como configurar o parâmetro N4 ou B4 dos Tags de Comunicação?

- Qual endereço utilizar no parâmetro *N4* ou *B4* de um Tag de Comunicação?

  - **Resposta**: Este endereço varia de equipamento para equipamento. Para saber qual o endereço exato a ser utilizado, consulte o manual de um equipamento ou entre em contato com o suporte técnico. O tópico **Dicas de Endereçamento (Modbus Convention)** deste manual contém algumas dicas de convenções comuns usadas por muitos fabricantes.

Quando utilizar os controles de RTS e DTR disponíveis na aba Serial da janela de configurações deste Driver?

- Um equipamento está comunicando diretamente na porta serial RS-232 de um computador. Como devo configurar os controles **RTS** e **DTR**?

  - **Resposta**: Deve-se consultar a documentação deste equipamento ou o suporte do fabricante para saber a configuração correta.

- Um equipamento está comunicando através de um conversor RS232-RS485 ligado à porta serial RS-232 de um computador. Como configurar os controles **RTS** e **DTR**?

  - **Resposta**: Na comunicação com equipamentos usando conversores RS232-RS485, tais configurações dependem do conversor. Este equipamento, ou **Escravo**, não tem influência, já que estes sinais só existem no lado serial RS-232, sem equivalentes no meio serial RS-485. O controle **RTS** é geralmente utilizado em conversores mais antigos para o chaveamento entre os modos de transmissão e recepção, pois o RS-485 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 e podem 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 por um equipamento não corresponde à ordem padrão do protocolo Modbus, onde os bytes mais significativos vêm sempre antes, ou padrão *big endian*, também chamado *Motorola*. Se este Driver está lendo valores absurdos ou diferentes dos valores armazenados em um equipamento, 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, os parâmetros *N* ou *B*, consulte o item **Byte Order** do tópico **Aba Operations**. Também recomenda-se consultar a documentação do equipamento.

- Uma aplicação está tentando ler um valor **Word**, porém este valor vem diferente do que está configurado neste equipamento. Se neste equipamento é configurado o valor "1" (um), nesta aplicação aparece o valor "256".

  - **Resposta**: O valor 1 (um) em hexadecimal é 0001H e o valor 256 em hexadecimal corresponde a 0100H. Este equipamento possui um *byte order* diferente do padrão do protocolo. Deve-se habilitar a opção **Swap Bytes**, a opção "b1" na **configuração por Strings**, para ler o valor correto.

- Uma aplicação tem um Tag configurado para ler um valor **DWord**, mas o valor lido por esta aplicação é diferente do valor armazenado neste equipamento. Ao atribuir o valor "258", por exemplo, ao registro neste equipamento, nesta aplicação aparece o valor absurdo "16908288".

  - **Resposta**: O valor 258 em hexadecimal é 00000102H e o valor 16908288 em hexadecimal corresponde a 01020000H. Este 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**, a opção "b2" na **configuração por Strings**, para ler o valor correto.

Como ler corretamente tipos de dados Float em equipamentos 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, os parâmetros *N* ou *B*, consulte o item **Byte Order** do tópico **Aba Operations**.

Particularidades conhecidas dos equipamentos da família ABB Advant Controller 31 Series 90, como por exemplo o ABB 07KT97

- Uma aplicação no **Elipse E3**, no **Elipse Power** ou no **Elipse Water** está tentando ler registros ou bits de um equipamento, mas sempre ocorrem erros.
  - **Resposta**: Os equipamentos desta série não permitem o uso de Superblocos do **Elipse E3**, do **Elipse Power** ou do **Elipse Water** pelos seguintes motivos:
    - Existem descontinuidades no mapa de endereços de registradores destes equipamentos, com intervalos de endereços não definidos.
    - O tamanho máximo do **PDU** é diferente do estabelecido pelo padrão do protocolo, e é definido como um tamanho que suporte 96 **Words** ou **Bits**. Uma vez que o protocolo agrupa 8 (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 por este 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** deste 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, consulte a seção sobre **Agrupamento Manual** no tópico **Leitura por Superblocos (Agrupamento)**.

NOTA

Pode ainda ser possível utilizar o agrupamento automático, ou Superblocos, caso não seja 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 opção **Customize Max. PDU Size** na aba **Modbus**, de acordo com o limite de 96 **Words**, 96 × 2 = 192 bytes, ou 96 **Bits**, 96 ÷ 8 = 12 bytes. Tal possibilidade pode ser avaliada com cuidado, caso a caso, pelo desenvolvedor de uma aplicação.

Uma aplicação está tentando ler valores do tipo de dados Float e a seguinte mensagem aparece no log deste 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 deste equipamento por que está retornando valores não normalizados.
- **Informações Adicionais**: Tal mensagem indica que este equipamento enviou a este Driver um valor de ponto flutuante, um **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 por exemplo as condições de *overflow*, *underflow*, $+\infty$ e $-\infty$, entre outras. Os valores não normalizados estão previstos na norma IEEE 754 e não deve, em tese, gerar problemas para este Driver ou para uma aplicação. Entretanto, devido à detecção de erros no passado relacionada a tipos específicos de equipamentos, este Driver passou a retornar 0 (zero) para uma aplicação ao receber valores não normalizados de um equipamento, registrando esta mensagem em log.

Lista de Equipamentos que Comunicam com o Protocolo 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 protocolo Modbus

FABRICANTE	EQUIPAMENTO
ABB	ETE30, MGE 144, KT97 e KT98
Altus	Praticamente todos os equipamentos da Altus possuem Modbus, exceto alguns modelos da linha Piccolo
Areva	MiCOM P127 e relé P632
Atos	Os equipamentos ATOS suportam o protocolo **Modbus RTU** com pequenas variações relativas ao tamanho máximo dos *frames* e *byte order*. Para mais informações sobre as variações, consulte o artigo *Utilizando o driver Modbus Master com controladores ATOS* no **Elipse Knowledgebase**
BCM	BCM1088, BCM1086, BCM-GP3000 e BCM2085
Ciber Brasil	Medidor multivariável de grandezas elétricas UDP200 e medidor multivariável de grandezas elétricas UDP600
Contemp	CPM45
Deep Sea	DSE5210, DSE5310, DSE5310M, DSE5320, DSE5510, DSE5510M, DSE5520, DSE7310 e DSE7320
Embrasul	Medidor MD4040
Eurotherm	2500 Intelligent Data Acquisition and Precision Multi-Loop PID Control
Fatek	FB - 14MCU
GE	GE PAC RX7 e GE GEDE UPS
Gefran	Equipamentos Gilogk II e 600RDR21
Hitachi	HDL17264
Honeywell	Equipamento HC-900
Horner APG	Equipamentos da série XLe/XLt all-in-one control devices
Koyo	Equipamento CPU 260
Kron	Medidor MKM-120
LG	DMT40U
LG / LSIS	LG Master K120 S e LG XGB - XBM
Moeller	XC100 - Porta Serial 232, XC200 - Porta Serial 232/422/485 (módulo de comunicação XIO-SER) e Porta Ethernet, XV200 - Porta Serial 232 e Porta Ethernet, XVH300 - Porta Serial 232 e Porta Ethernet e XV400 - Porta Serial 232 e Porta Ethernet

FABRICANTE	EQUIPAMENTO
Novus	N1100, N1500, N2000 e N3000
Schneider	Twido, A340, M340, série Premium, conversores de frequência e *soft starter*, disjuntores de MT e BT, relés de proteção de BT e MT e relés SEPAM das séries 20, 40 e 80
Telemecanique	Controladores Zelio Logic com final "BD"
Unitronics	V120
Weg	TP 03 e Clic 02
Yaskawa	V1000

Lista de Exceções Padrão do Protocolo Modbus

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

As exceções são registradas no log deste Driver, sempre que detectadas, e podem ser lidas por uma aplicação através da **leitura do código da última exceção**.

Note que, além das exceções listadas aqui, um equipamento pode definir outras exceções proprietárias. Neste caso, espera-se que estas exceções sejam descritas na documentação do fabricante deste 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 esta função não está implementada ou que um **Escravo** encontra-se em um estado inadequado para processá-la
02	ILLEGAL DATA ADDRESS	O endereço de dados recebido não é um endereço válido. Mais especificamente, a combinação de um endereço de referência e a quantidade de dados a serem transferidos é inválida
03	ILLEGAL DATA VALUE	O valor presente na requisição de um **Mestre** não é válido. Isto indica uma falha na estrutura de dados remanescente de uma requisição complexa, como quando o tamanho informado para um bloco de dados não está correto. Esta exceção não indica que os valores submetidos para escrita estejam fora do escopo esperado por uma aplicação, uma vez que esta informação não é acessível ao protocolo
04	SLAVE DEVICE FAILURE	Ocorreu um erro irrecuperável durante o processamento da função solicitada
05	ACKNOWLEDGE	Usado com comandos de programação. Um **Escravo** aceitou a mensagem e está processando, mas este processamento demanda um

CÓDIGO (EM HEXADECIMAL)	NOME	SIGNIFICADO
		longo tempo. Esta exceção previne um *time-out* em um **Mestre**. O fim desta requisição deve ser testado por um processo de *polling*
06	SLAVE DEVICE BUSY	Usado com comandos de programação. Indica que um **Escravo** está processando um outro comando de longa duração e que esta solicitação deve ser retransmitida mais tarde, quando este **Escravo** está novamente disponível
08	MEMORY PARITY ERROR	Usado em conjunto com as funções **20** e **21**, *reference type 6*, para indicar que a área estendida de arquivos falhou em um teste de consistência. Um equipamento **Escravo** pode estar precisando de manutenção
0A	GATEWAY PATH UNAVAILABLE	Usado em conjunto com *gateways* para indicar que um *gateway* não foi capaz de alocar um caminho interno para o processamento de uma solicitação. Geralmente indica que um *gateway* está desconfigurado ou sobrecarregado
0B	GATEWAY TARGET DEVICE FAILED TO RESPOND	Usado em conjunto com *gateways* para indicar que não foi recebida nenhuma resposta de um equipamento de destino. Geralmente indica que este equipamento não está presente na rede

Limite Máximo para o Tamanho dos Blocos Suportado pelo Protocolo Modbus

Neste tópico são apresentados os limites máximos de tamanho de bloco suportados pelo protocolo Modbus, na atual versão **1.1b** da especificação. Para mais informações, consulte a especificação no *site oficial do protocolo*.

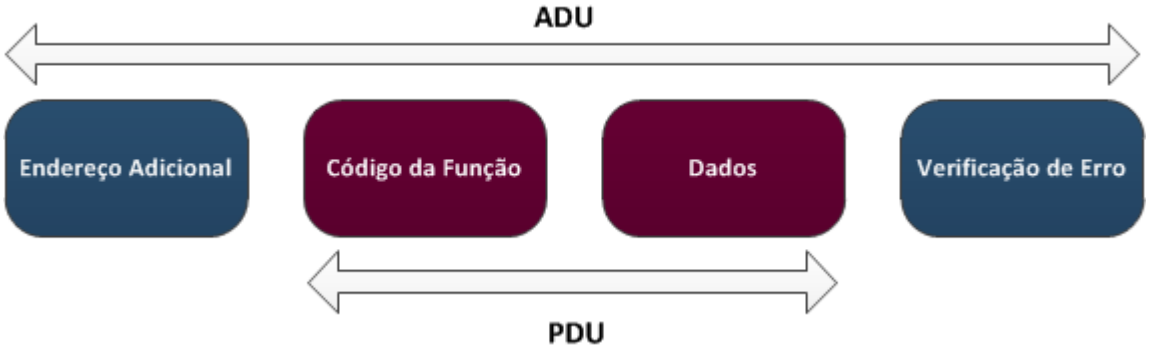
Note que, devido aos recursos de **Leitura por Superblocos** e **Partição Automática de Blocos**, presentes na versão atual deste Driver, dificilmente o usuário necessita levar em conta estes limites em uma aplicação, uma vez que este Driver já realiza automaticamente as otimizações necessárias no momento 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 de um equipamento, caso seja obrigado a realizar o agrupamento de forma manual. Para mais informações, consulte o tópico **Leitura por Superblocos (Agrupamento)**. 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**, ou *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**, ou *Application Data Unit*.



Limites suportados pelo protocolo Modbus

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

Portanto, dependendo do tipo de dado ou função Modbus que é utilizado na comunicação, o protocolo impõe os limites de Elementos de 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 (*Holding Registers*)	123 registros (247 bytes)
01, 02	Leitura de múltiplos bits	2000 bits (250 bytes)
15	Escrita de múltiplos bits	1968 bits (247 bytes)
20	Leitura de registros de arquivo	124 registros (248 bytes)
21	Escrita de registros de arquivo	122 registros (244 bytes)

Para mais informações, consulte o *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 os valores em binário correspondentes na codificação BCD.

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 2 (dois) algarismos por byte. Note que, na tabela anterior, cada dígito decimal requer apenas 4 (quatro) bits, ou meio byte, para a representação.

Esta representação com 2 (dois) dígitos em cada byte é chamada de **BCD Comprimido**, ou *Packed BCD*, e é a representação utilizada por este Driver, ou seja, os pacotes enviados por este Driver com valores BCD utilizam 1 (um) byte de dados para cada 2 (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 1 (um) byte, o formato usado por este Driver. O primeiro passo é separar os 2 (dois) dígitos decimais que compõem o valor na representação decimal:

- **Dígito 1**: 8 (oito)
- **Dígito 2**: 4 (quatro)

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

Usando o formato BCD comprimido, entretanto, representa-se os 2 (dois) dígitos decimais separadamente em cada metade, ou *nibble*, do byte a ser enviado:

- **BCD**: 0x84 ou 10000100b

Note que, ao interpretar por engano este valor 0x84 em formato BCD como um valor em formato hexadecimal sem esta codificação, e este valor seja convertido para decimal, obtêm-se o valor 132, sem significado algum.

A tabela a seguir apresenta mais alguns exemplos de valores decimais entre 0 (zero) e 99 e as respectivas representações no formato BCD comprimido em 1 (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

DECIMAL	HEXADECIMAL	BCD (HEXADECIMAL)	BCD (BINÁRIO)
45	0x2D	0x45	01000101b

Exemplo de Configuração para o Formato BCD

- **Leitura do valor 11223344 em formato BCD**: Note que o inteiro 11223344 possui 8 (oito) algarismos, portanto, na janela de propriedades deste Driver deve-se criar uma nova Operação, definindo o campo **Data** com o tipo de dados **BCD** e o campo **Size** com tamanho 4 (quatro), ou seja, os 8 (oito) algarismos do valor BCD devem ser dimensionados para bytes, agrupando os algarismos sempre aos pares, 11 22 33 44, para saber quantos bytes são necessários para representar este valor

Leitura de um valor com codificação BCD comprimido

MEMÓRIA DE UM EQUIPAMENTO	VALOR DE UM TAG	NOTAS
Registro 1 = 0x1122 e Registro 2 = 0x3344	11223344	O valor 11223344 necessita de 4 (quatro) bytes para ser representado em formato **BCD**, e é necessário ocupar 2 (dois) registros Modbus na memória de um equipamento

Documentação das Interfaces de Comunicação

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

Configurações de um Driver

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

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

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

1. Abra o Organizer.
2. Selecione um 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, no caso de acesso a duas portas seriais, é preciso adicionar dois Drivers em um aplicação e configurar cada um destes Drivers para cada porta serial.

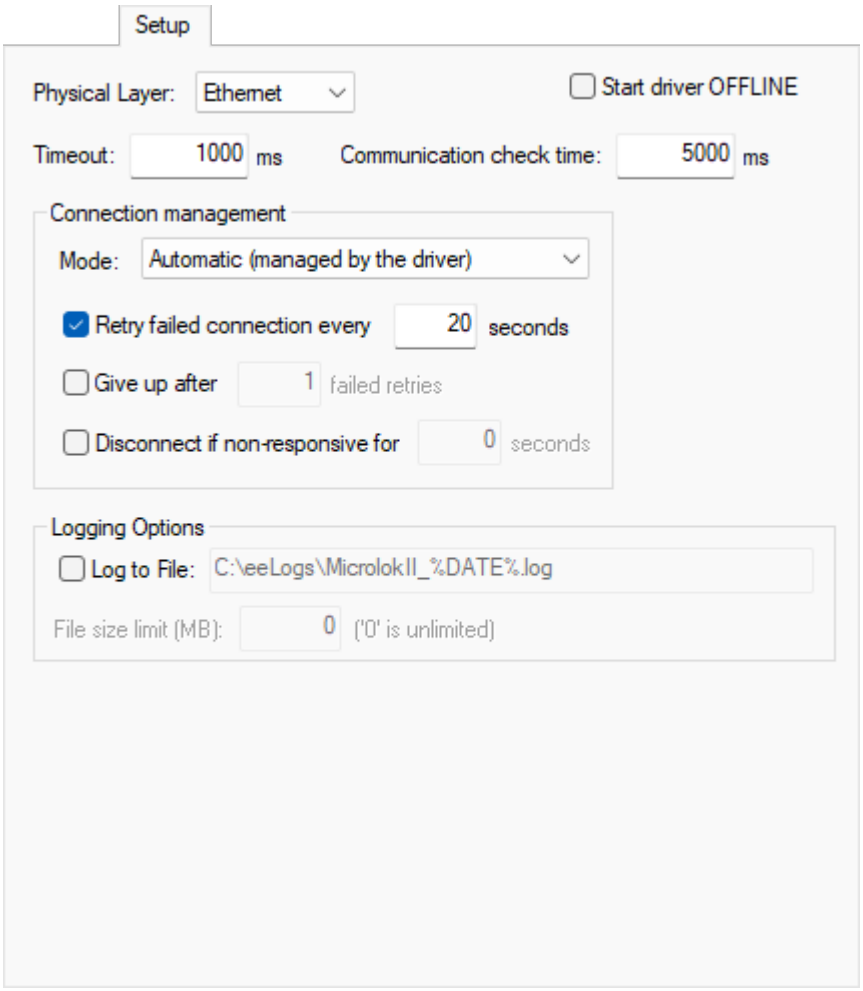
Caixa de Diálogo de Configuração

A caixa de diálogo das Interfaces de Configuração permite configurar a conexão de I/O que é utilizada por um 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 aquele Driver, na caixa de diálogo de configuração.

Aba Setup

A aba **Setup** contém a configuração geral de um Driver. Esta aba é dividida nos seguintes grupos:

- **Configurações gerais**: Configurações da camada física de um 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



Aba Setup

Opções gerais da aba Setup

OPÇÃO	DESCRIÇÃO
Physical Layer	Selecione a interface física em uma lista. As opções disponíveis são **Serial**, **Ethernet**, **Modem** e **RAS**. A interface selecionada deve ser configurada na aba específica
Timeout	Configure o *time-out*, em milissegundos, para a camada física. Esta é a medida de tempo que a interface de I/O

OPÇÃO	DESCRIÇÃO
	aguarda para a recepção de um byte qualquer do *buffer* de recepção
Communication check time	Configure o tempo, em milissegundos, para definir o intervalo em que a comunicação é considerada em estado inativo. Enquanto um Driver de Comunicação receber dados válidos, o estado de comunicação é considerado ativo. Porém, se durante o funcionamento um Driver de Comunicação não receber dados válidos neste período de tempo, o estado é considerado inativo. O estado de comunicação é mostrado no Tag **IO.CommunicationStatus**
Start driver OFFLINE	Selecione esta opção para que um Driver inicie em modo **Offline** ou parado. Isto significa que a interface de I/O não é criada até que se configure um Driver em modo **Online** utilizando-se um Tag em uma aplicação. Este modo possibilita a configuração dinâmica da interface de I/O em tempo de execução

Opções para o grupo Connection management

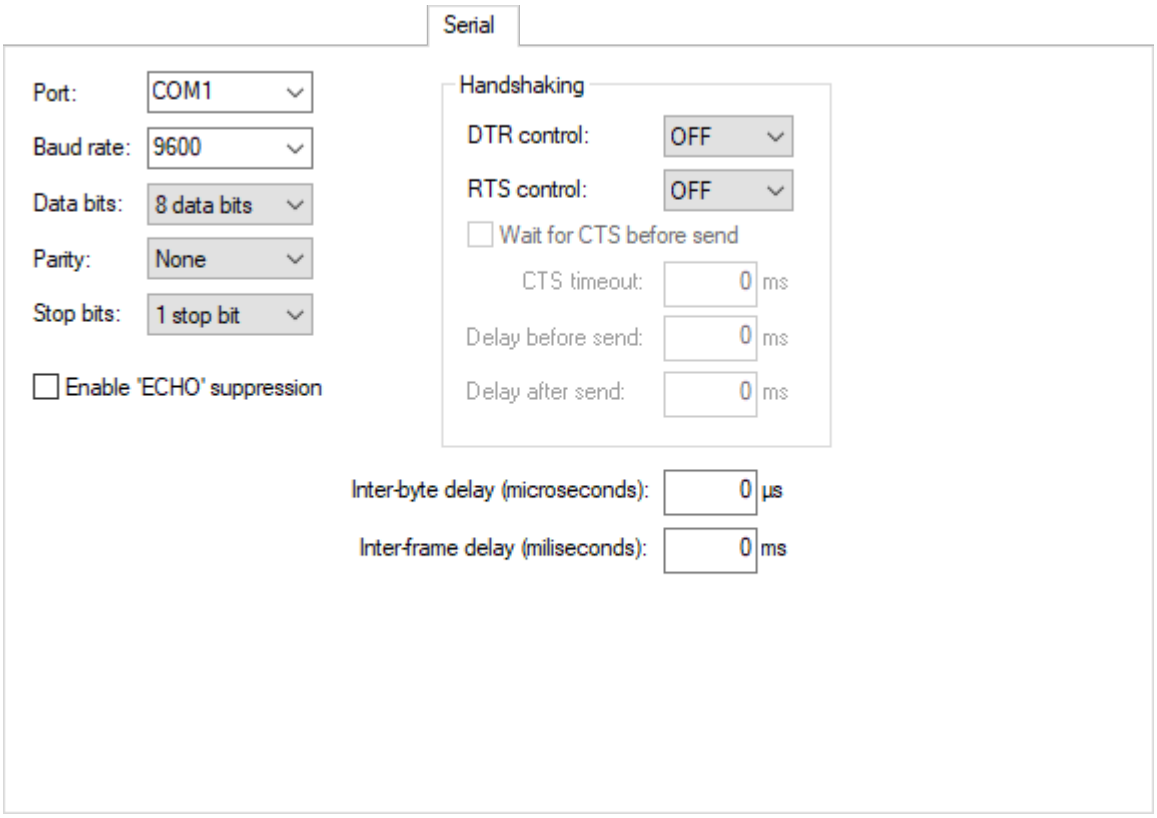
OPÇÃO	DESCRIÇÃO
Mode	Seleciona o modo de gerenciamento de conexão. Selecionar a opção **Automatic** permite que um Driver gerencie a conexão automaticamente, como especificado nas opções seguintes. Selecionar a opção **Manual** permite que uma aplicação gerencie a conexão completamente
Retry failed connection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

Opções para o grupo Logging Options

OPÇÃO	DESCRIÇÃO
Log to File	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. Caso se utilize a macro **%PROCESS%** no nome do arquivo de log, esta é substituída pelo identificador do processo atual. Esta opção é particularmente útil ao se utilizar várias instâncias de um mesmo Driver no **Elipse E3**, permitindo assim que cada instância gere um arquivo separado de log. Por exemplo, ao configurar esta opção com o valor "c:\e3logs\drivers\sim_%PROCESS%.log", gera-se um arquivo c:\e3logs\drivers\sim_00000FDA.log para o processo **0FDAh**. Pode-se também utilizar a macro **%DATE%** no nome do arquivo. Neste caso é gerado um arquivo de log por dia, no formato **aaaa_mm_dd**. Por exemplo, ao configurar esta opção com o valor "c:\e3logs\drivers\sim_%DATE%.log", gera-se o arquivo c:\e3logs\drivers\sim_2005_12_31.log em 31/12/2005 e o arquivo c:\e3logs\drivers\sim_2006_01_01.log em 01/01/2006. De forma semelhante, a macro **%DATE_HOUR%** gera um arquivo de log por hora, no formato **aaaa_mm_dd_hh**
File size limit (MB)	Configure o limite de tamanho do arquivo de log, em megabytes. Um valor igual a 0 (zero) significa que não há limite de tamanho para o arquivo de log

Aba Serial

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



Aba Serial

Opções gerais da aba Serial

OPÇÃO	DESCRIÇÃO
Port	Selecione uma porta serial a partir da lista, de **COM1** até **COM4**, ou digite o nome de uma porta serial no formato **COM*n***, como por exemplo "COM15". Ao digitar o nome de uma porta serial manualmente, a caixa de diálogo aceita apenas nomes de portas seriais começando com a expressão "COM"
Baud rate	Selecione um *baud rate* a partir da lista (**1200**, **2400**, **4800**, **9600**, **19200**, **38400**, **57600** ou **115200**) ou digite um *baud rate*, como por exemplo 600
Data bits	Selecione 7 (sete) ou 8 (oito) bits de dados a partir da lista
Parity	Selecione uma paridade a partir da lista. As opções disponíveis são **None**, **Even**, **Odd**, **Mark** ou **List**
Stop bits	Selecione o número de stop bits a partir da lista. As opções disponíveis são **1**, **1.5** ou **2** stop bits
Enable 'ECHO' supression	Habilite esta opção para remover o eco recebido após a Interface de Comunicação enviar dados por uma porta serial. Se o eco não é igual aos bytes recém enviados, a Interface de Comunicação aborta a comunicação
Inter-byte delay (microseconds)	Defina uma espera entre cada byte transmitido pela Interface de Comunicação, em milionésimos de segundo, ou seja, 1000000 é igual a um segundo. Esta opção deve ser utilizada com esperas pequenas de menos de um milissegundo
Inter-frame delay (milliseconds)	Defina uma espera entre pacotes enviados ou recebidos pela Interface de Comunicação, em milésimos de segundo,

OPÇÃO	DESCRIÇÃO
	ou seja, 1000 é 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

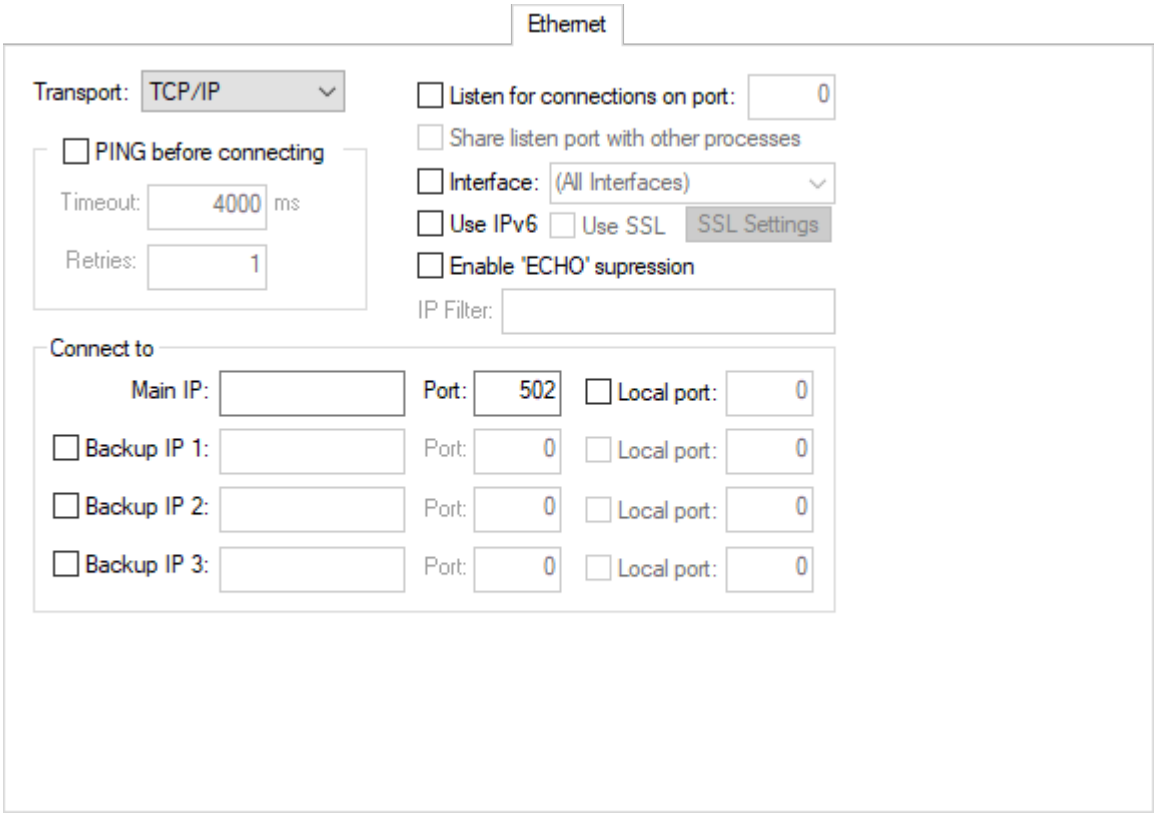
O grupo **Handshaking** configura o uso dos sinais **RTS**, **CTS** e **DTR** no processo de *handshaking* ou seja, controla quando um dado pode ser enviado ou recebido através de uma linha serial. Na maioria das vezes, configurar a opção **DTR control** para **ON** e a opção **RTS control** para **Toggle**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

Opções disponíveis no grupo Handshaking

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

Aba Ethernet

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



Aba Ethernet

Opções disponíveis na aba Ethernet

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

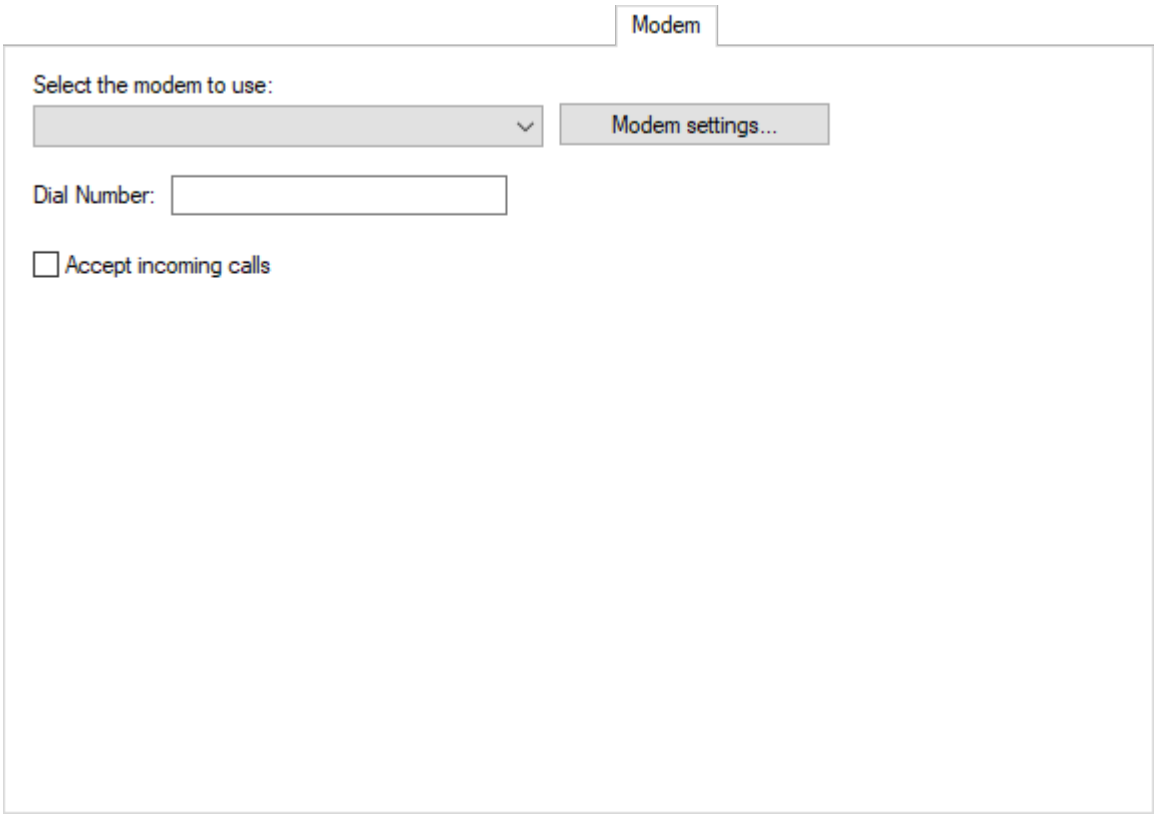
OPÇÃO	DESCRIÇÃO
IP Filter	Lista de endereços IP restringidos ou permitidos de onde um Driver aceita conexões (*Firewall*). Consulte a propriedade **IO.Ethernet.IPFilter** para mais informações
PING before connecting	Habilite esta opção para executar um comando **ping**, ou seja, para verificar se um dispositivo pode ser encontrado na rede, em um dispositivo antes de tentar uma conexão com o *socket*. Esta é uma maneira rápida de determinar uma conexão bem sucedida antes de tentar abrir um *socket* com um dispositivo. O *time-out* de uma conexão com um *socket* pode ser bem alto. As opções disponíveis são: • **Timeout**: Especifique o número de milissegundos de espera por uma resposta de um comando **ping**. Deve-se usar um comando **ping** 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, ou seja entre 1 (um) e 4 (quatro) segundos • **Retries**: Número de retentativas de um comando **ping**, sem contar a tentativa inicial. Se todas as tentativas falharem, então a conexão com o *socket* é abortada

Opções disponíveis no grupo Connect to

OPÇÃO	DESCRIÇÃO
Main IP	Digite o endereço IP de um dispositivo remoto. Pode-se usar tanto o endereço IP separado por pontos quanto uma URL. No caso de uma URL, um Driver usa o serviço de DNS disponível para mapear a URL para um endereço IP, como por exemplo "192.168.0.13" ou "Server1"
Port	Digite a porta IP de um dispositivo remoto, entre 0 (zero) e 65535
Local port	Selecione esta opção para utilizar uma porta IP local fixa ao conectar a um dispositivo remoto
Backup IP 1, 2 e 3	Indique o endereço IP, a porta IP e a porta IP local fixa de até 3 (três) endereços de *backup* de um dispositivo remoto

Aba Modem

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

Aba RAS

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

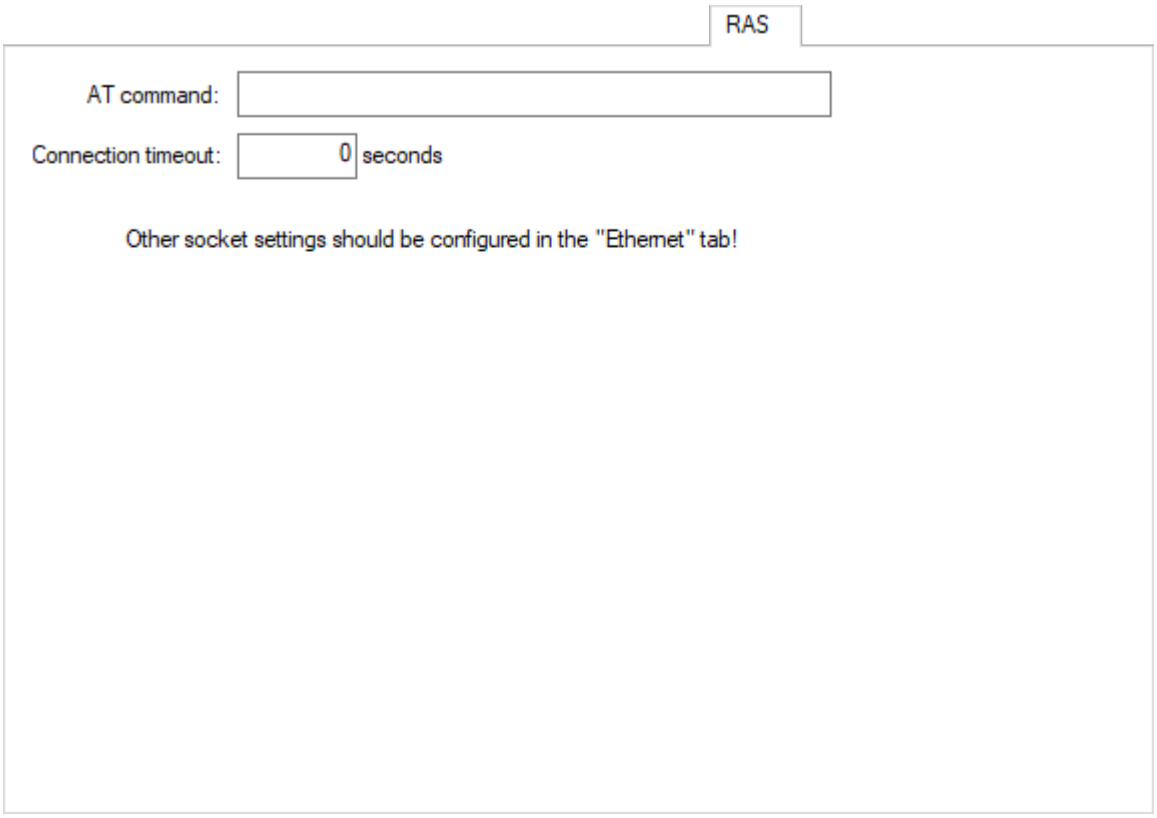
A Interface **RAS** abre uma conexão *socket* com um dispositivo RAS. Um dispositivo RAS é um servidor de modems acessível através de TCP/IP, aguardando por conexões *socket* em uma porta IP. Para cada conexão aceita nesta porta tem-se acesso a um modem.

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 passos de inicialização ou de conexão a seguir são efetuados:

1. Limpeza do *socket*, ou seja, remove qualquer mensagem de saudação **TELNET** recebida de um dispositivo RAS.
2. Envio de um comando de discagem **AT**, no formato **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 um dispositivo, ou seja, a conexão foi estabelecida.

Se o passo 5 (cinco) é efetuado com sucesso, então o *socket* comporta-se como um *socket* normal, com o dispositivo RAS funcionando como um roteador entre um Driver e o dispositivo. Os bytes enviados por um 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 a um Driver utilizando o mesmo *socket*.

Depois que a conexão é estabelecida, a Interface **RAS** monitora os dados recebidos por um 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**.



Aba RAS

Opções disponíveis na aba RAS

OPÇÃO	DESCRIÇÃO
AT command	Uma **String** com o comando **AT** completo usado para discar para um dispositivo de destino. Por exemplo, "ATDT33313456" disca por tom para o número "33313456"
Connection timeout	Número de segundos a aguardar por uma resposta **CONNECT** do modem, após o envio de um comando **AT**

Configurações Gerais

Esta seção contém informações sobre a configuração dos **Tags de Comunicação** e das **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.CommunicationStatus

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Leitura
Parâmetro N1	-1 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	0 (zero)
Parâmetro N4	6 (seis)
Configuração por String	IO.CommunicationStatus

Este Tag informa o estado da comunicação de um Driver. Indica o funcionamento da comunicação em função do recebimento de dados válidos dentro de um período de tempo arbitrado na configuração. Para mais informações, consulte o tópico **Aba Setup**. Os valores possíveis são **0 - Comunicação inativa**: O Driver não recebeu dados válidos ou deixou de receber dados depois de *n* milissegundos, conforme configurado na janela de propriedades, ou **1 - Comunicação ativa**: O Driver está recebendo dados válidos.

IO.IOKitEvent

Tipo de Tag	Tag Bloco
Tipo de Acesso	Somente Leitura
Parâmetro B1	-1 (menos um)
Parâmetro B2	0 (zero)
Parâmetro B3	0 (zero)
Parâmetro B4	1 (um)
Propriedade Size	4 (quatro)
Propriedade ParamItem	IO.IOKitEvent

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

- **Elemento 0**: Tipo de evento. Os valores possíveis são **0**: Informação, **1**: Advertência ou **2**: Erro
- **Elemento 1**: Fonte de um evento. Os valores possíveis são **0**: Driver (específico de um Driver), **-1**: IOKit (eventos genéricos da Interface de Comunicação), **-2**: Interface **Serial**, **-3**: Interface **Modem**, **-4**: Interface **Ethernet** ou **-5**: Interface **RAS**
- **Elemento 2**: Número do erro, específico de cada fonte de evento
- **Elemento 3**: Mensagem de um evento, uma **String** específica de cada evento

NOTA

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

IO.PhysicalLayerStatus

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Leitura
Parâmetro N1	-1 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	0 (zero)
Parâmetro N4	2 (dois)
Configuração por String	IO.PhysicalLayerStatus

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

- **0**: Camada física parada, ou seja, um 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, ou seja, um Driver está em modo **Online**, mas a camada física não está conectada. Se a opção **Connection management** está configurada com o valor **Automatic**, a camada física pode

estar conectando, desconectando ou esperando por uma tentativa de reconexão. Se a opção **Connection management** está configurada com o valor **Manual**, então a camada física permanece neste estado até ser forçada a conectar

- **2**: Camada física conectada, ou seja, a camada física está pronta para ser usada. Isto **NÃO** significa que um equipamento esteja conectado, apenas que a camada de acesso está funcionando

IO.SetConfigurationParameters

Tipo de Tag	Tag Bloco
Tipo de Acesso	Somente Leitura
Parâmetro B1	-1 (menos um)
Parâmetro B2	0 (zero)
Parâmetro B3	0 (zero)
Parâmetro B4	3 (três)
Propriedade Size	2 (dois)
Propriedade ParamItem	IO.SetConfigurationParameters

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

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

No **Elipse SCADA** é necessário usar um Tag Bloco. Cada parâmetro a ser configurado utiliza dois Elementos de Bloco. Por exemplo, caso seja necessário configurar 3 (três) parâmetros, então o tamanho do Bloco deve ser 6 (seis, 3 × 2). O primeiro Elemento é o nome da propriedade, como uma **String**, e o segundo Elemento é o valor desta propriedade, conforme o exemplo a seguir.

```
// 'Block' deve ser um Tag Bloco 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 Bloco inteiro
Block.Write()
```

Ao usar o **Elipse 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** de um Driver para enviar os parâmetros diretamente para este Driver, sem a necessidade de criar um Tag, conforme o exemplo a seguir.

```
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 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
```

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

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Leitura ou Escrita
Parâmetro N1	-1 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	0 (zero)
Parâmetro N4	4 (quatro)
Configuração por String	IO.WorkOnline

Este Tag informa o estado atual de um Driver e permite iniciar ou parar a camada física. Os valores possíveis são os seguintes:

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

No exemplo a seguir, utilizando o **Elipse E3**, um Driver é colocado em modo **Offline**, a 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 ao configurar um Driver em modo **Online**, ou seja, escrevendo o valor 1 (um). Neste caso, este 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**
- Este Driver pode ter ficado sem memória
- A camada física pode ter deixado de criar a *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 ou falha ao inicializar o TAPI (modem), entre outras. A causa é gravada no arquivo de log

IMPORTANTE

Mesmo que a configuração de um Driver para o modo **Online** seja bem-sucedida, isto não significa necessariamente que a camada física esteja pronta para uso, ou seja, 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. Os valores possíveis são **0**: Modo automático, em que um Driver gerencia a conexão ou **1**: Modo manual, em que uma aplicação gerencia a conexão.

IO.GiveUpEnable

◩ Quando configurada para Verdadeiro, define um número máximo de tentativas de reconexão. Se todas as reconexões falharem, um Driver entra em modo **Offline**. Se configurada para Falso, um 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), um Driver tenta apenas uma reconexão quando a conexão é perdida. Se esta falhar, este 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 está 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

Número de segundos para a verificação de inatividade. Se a camada física está inativa por este período de tempo, então é desconectada.

IO.RecoverEnable

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

IO.RecoverPeriodSec

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 um Driver em modo **Offline** e em Falso para iniciar um Driver em modo **Online**.

NOTA

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

IO.TimeoutMs

Define o *time-out* da camada física, em milissegundos. Um segundo equivale a 1000 milissegundos.

IO.Type

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

- **N ou None**: Não utiliza uma interface física, ou seja, um Driver deve fornecer uma interface personalizada
- **S ou Serial**: Utiliza uma porta serial local (COM*n*)
- **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*). Um Driver conecta-se a um 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 das **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 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	0 (zero)
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

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Leitura
Parâmetro N1	-1 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	0 (zero)
Parâmetro N4	1100
Configuração por String	IO.Stats.Partial.BytesSent

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

IO.Stats.Partial.TimeConnectedSeconds

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Leitura
Parâmetro N1	-1 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	0 (zero)
Parâmetro N4	1102
Configuração por String	IO.Stats.Partial.TimeConnectedSeconds

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

IO.Stats.Partial.TimeDisconnectedSeconds

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Leitura
Parâmetro N1	-1 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	0 (zero)
Parâmetro N4	1103
Configuração por String	IO.Stats.Partial.TimeDisconnectedSeconds

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

IO.Stats.Total.BytesRecv

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Leitura
Parâmetro N1	-1 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	0 (zero)
Parâmetro N4	1001
Configuração por String	IO.Stats.Total.BytesRecv

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

IO.Stats.Total.BytesSent

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Leitura
Parâmetro N1	-1 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	0 (zero)
Parâmetro N4	1000
Configuração por String	IO.Stats.Total.BytesSent

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

IO.Stats.Total.ConnectionCount

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Leitura
Parâmetro N1	-1 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	0 (zero)
Parâmetro N4	1004
Configuração por String	IO.Stats.Total.ConnectionCount

Este Tag retorna a quantidade de conexões que um 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 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	0 (zero)
Parâmetro N4	1002
Configuração por String	IO.Stats.Total.TimeConnectedSeconds

Este Tag retorna o número de segundos que um 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 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	0 (zero)
Parâmetro N4	1003
Configuração por String	IO.Stats.Total.TimeDisconnectedSeconds

Este Tag retorna o número de segundos que um 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 das **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 e também são válidos quando a Interface **RAS** está selecionada.

IMPORTANTE
Estes Tags estão disponíveis **SOMENTE** enquanto um Driver está em modo **Online**.

IO.Ethernet.IPSelect

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

Indica o endereço IP ativo. Os valores possíveis são **0**: O endereço principal de IP está selecionado, **1**: O primeiro endereço IP alternativo ou de *backup* está selecionado, **2**: O segundo endereço IP alternativo ou de *backup* está selecionado ou **3**: O terceiro endereço IP alternativo ou de *backup* está selecionado.

Se a Interface **Ethernet** ou **RAS** está conectada, este Tag indica qual dos quatro endereços IP configurados está em uso. Se a Interface está 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 está 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 um Driver está atualmente conectado. Se um Driver está desconectado, este Tag configura o endereço IP ativo para a próxima tentativa de conexão.

IO.Ethernet.IPSwitch

Tipo de Tag	Tag de Comunicação

Tipo de Acesso	Somente Escrita
Parâmetro N1	-1 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	4 (quatro)
Parâmetro N4	1 (um)
Configuração por String	IO.Ethernet.IPSwitch

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

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

IO.Ethernet.SocketState

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Leitura
Parâmetro N1	-1 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	4 (quatro)
Parâmetro N4	2 (dois)
Configuração por String	IO.Ethernet.SocketState

A propriedade **Value** deste Tag corresponde a estados do *socket* em um mapa de bits:

- **Bit 0**: 0 (zero, não está em escuta) ou 1 (um, em escuta)
- **Bit 1**: 0 (zero, desconectado) ou 1 (um, conectado)

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 um Driver não deve aceitar conexões externas, ou seja, se um Driver se comporta como mestre, ou configure em Verdadeiro para habilitar a recepção de conexões, ou seja, se um Driver se comporta como escravo.

IO.Ethernet.BackupEnable[2,3]

◩ Configure em Verdadeiro para habilitar o endereço IP alternativo ou de *backup*. Se a tentativa de reconectar com o endereço IP principal falhar, um Driver tenta utilizar um endereço IP alternativo ou de *backup*. Configure em Falso para

desabilitar a utilização.

IO.Ethernet.BackupIP[2,3]

Endereço IP alternativo ou de *backup* de um equipamento remoto. Pode-se utilizar tanto o endereço numérico como o nome de *host* de um equipamento, como por exemplo "192.168.0.7" ou "SERVER2".

IO.Ethernet.BackupLocalPort[2,3]

Número da porta local a ser utilizada na conexão ao endereço IP alternativo ou de *backup* de um equipamento remoto. Usado apenas se a propriedade **IO.Ethernet.BackupLocalPortEnable** está configurada para Verdadeiro.

IO.Ethernet.BackupLocalPortEnable[2,3]

Configure em Verdadeiro para forçar o uso de uma porta local específica ao conectar ao endereço IP alternativo ou de *backup* ou configure em Falso para utilizar qualquer porta local disponível.

IO.Ethernet.BackupPort[2,3]

Número da porta do endereço IP alternativo ou de *backup* de um equipamento remoto, usado juntamente com a propriedade **IO.Ethernet.BackupIP**.

IO.Ethernet.IPFilter

Lista de endereços IPv4 ou IPv6 separados por vírgula, que define de quais endereços um 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, conforme os exemplos a seguir:

- **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:*:*)**: 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 um 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 corresponde 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 não 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 não encontrados na lista são autorizados

Se um endereço IP aparece 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, a mensagem "Blocked incoming socket connection from {IP}!" é logada.

No caso de conexões UDP em modo escuta em *broadcast*, em que um 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, a mensagem "Blocked incoming packet from {IP} (discarding {N} bytes)!" é logada.

IO.Ethernet.ListenIP

A Endereço IP da interface local de rede por onde um 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 por um Driver para escutar conexões.

IO.Ethernet.MainIP

A Endereço IP de um equipamento remoto. Pode-se utilizar tanto o endereço numérico como o nome de *host* de um 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 de um equipamento remoto. 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 ou configure em Falso para utilizar qualquer porta local disponível.

IO.Ethernet.MainPort

9 Número da porta IP em um equipamento remoto, usado em conjunto com a propriedade **IO.Ethernet.MainIP**.

IO.Ethernet.PingEnable

◩ Configure em Verdadeiro para habilitar o envio de um comando **ping** para o endereço IP de um equipamento remoto, 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.SupressEcho

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 um Driver enviou para um equipamento.

IO.Ethernet.Transport

Define o protocolo de transporte. Os valores possíveis são **T ou TCP**: Utiliza o protocolo TCP/IP ou **U ou UDP**: Utiliza o protocolo UDP/IP.

IO.Ethernet.UseIPv6

Configure em Verdadeiro para utilizar endereços IPv6 em todas as conexões Ethernet ou configure em Falso para utilizar endereços IPv4 (padrão).

Configuração da Interface Modem

Esta seção contém informações sobre a configuração dos **Tags de Comunicação** e das **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 um Driver está em modo **Online**.

IO.TAPI.ConnectionBaudRate

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Leitura
Parâmetro N1	-1 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	3 (três)
Parâmetro N4	5 (cinco)
Configuração por String	IO.TAPI.ConnectionBaudRate

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

IO.TAPI.Dial

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Escrita
Parâmetro N1	-1 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	3 (três)
Parâmetro N4	1 (um)
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 uma chamada é estabelecida.

IO.TAPI.HangUp

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Escrita
Parâmetro N1	-1 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	3 (três)
Parâmetro N4	4 (quatro)
Configuração por String	IO.TAPI.HangUp

Qualquer valor escrito neste Tag desliga a chamada atual.

NOTA

Use este comando apenas quando gerenciar a camada física manualmente ou ao explicitamente tentar forçar um Driver a reiniciar a comunicação. Se a camada física está configurada para reconexão automática, um Driver imediatamente tenta restabelecer a conexão.

IO.TAPI.IsModemConnected

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Leitura
Parâmetro N1	-1 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	3 (três)
Parâmetro N4	3 (três)
Configuração por String	IO.TAPI.IsModemConnected

Este Tag indica o estado da conexão do modem. Os valores possíveis são **0**: O modem não está conectado, mas pode estar realizando ou recebendo uma chamada externa ou **1**: O modem está conectado e um Driver completou ou recebeu uma chamada externa com sucesso. Enquanto está 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 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	3 (três)
Parâmetro N4	6 (seis)
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 **0**: O modem não está conectado, **1**: O modem está conectando, ou seja, realizando ou recebendo uma chamada externa, **2**: O modem está conectado. Enquanto está neste estado, a camada física consegue enviar ou receber dados ou **3**: O modem está desconectando a chamada atual.

IO.TAPI.ModemStatus

Tipo de Tag	Tag de Comunicação
Tipo de Acesso	Somente Leitura
Parâmetro N1	-1 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	3 (três)
Parâmetro N4	2 (dois)
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!"**: Um Driver não conseguiu inicializar a linha do modem. Confira o arquivo de log deste Driver para mais detalhes
- **"Modem error at dial!"**: Um Driver não conseguiu começar ou aceitar uma chamada
- **"Connecting..."**: Um 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!"**: Um Driver conectou-se com sucesso, ou seja, completou ou aceitou uma chamada externa
- **"Disconnecting..."**: Um Driver está desligando a chamada atual
- **"Disconnected OK!"**: Um Driver desligou a chamada atual
- **"Error: no dial tone!"**: Um Driver abortou a chamada porque o sinal de linha disponível não foi detectado
- **"Error: busy!"**: Um Driver abortou a ligação porque a linha estava ocupada
- **"Error: no answer!"**: Um 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 (menos um)
Parâmetro N2	0 (zero)
Parâmetro N3	3 (três)
Parâmetro N4	0 (zero)
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**.

Propriedades

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

IO.TAPI.AcceptIncoming

Configure em Falso se o modem não pode aceitar chamadas externas, ou seja, se um Driver se comporta como mestre, e configure em Verdadeiro para habilitar a recepção de chamadas, ou seja, se um 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 um Driver deve utilizar o primeiro modem disponível.

IO.TAPI.PhoneNumber

A O número de telefone utilizado em comandos **Dial**, como por exemplo "0w01234566", em que 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 das **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

A Comando **AT** a ser enviado através do *socket* para forçar um equipamento RAS a realizar uma ligação usando o canal RAS atual, como por exemplo "ATDT6265545".

IO.RAS.CommandTimeoutSec

9 Tempo de espera pela mensagem **CONNECT** em resposta a um comando **AT**, em segundos.

Configuração da Interface Serial

Esta seção contém informações sobre a configuração dos **Tags de Comunicação** e das **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 expira, um Driver aborta o envio de bytes através da porta serial. Disponível apenas quando a propriedade **IO.Serial.RTS** está configurada com o valor **Toggle** e a propriedade **IO.Serial.WaitCTS** está configurada em Verdadeiro.

IO.Serial.DataBits

Especifica o número de bits de dados para a configuração da porta serial. Os valores possíveis são **5**: Cinco bits de dados, **6**: Seis bits de dados, **7**: Sete bits de dados ou **8**: Oito bits de dados.

IO.Serial.DelayAfterMs

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 com o valor **Toggle** e a propriedade **IO.Serial.WaitCTS** está configurada em Falso.

IO.Serial.DelayBeforeMs

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 com o valor **Toggle** e a propriedade **IO.Serial.WaitCTS** está configurada em Falso.

IO.Serial.DTR

Indica o modo como um Driver lida com o sinal **DTR**. Os valores possíveis são **OFF**: Sinal **DTR** sempre desligado ou **ON**: Sinal **DTR** sempre ligado.

IO.Serial.InterbyteDelayUs

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

IO.Serial.InterframeDelayMs

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

IO.Serial.Parity

Especifica a paridade para a configuração da porta serial. Os valores possíveis são **E ou Even**: Paridade par, **N ou None**: Sem paridade, **O ou Odd**: Paridade ímpar, **M ou Mark**: Paridade de marca ou **S ou Space**: Paridade de espaço.

IO.Serial.Port

Número da porta serial local. Os valores possíveis são **1**: Utiliza a porta COM1, **2**: Utiliza a porta COM2, **3**: Utiliza a porta COM3 ou **n**: Utiliza a porta COM*n*.

IO.Serial.RTS

Indica como um Driver lida com o sinal **RTS**. Os valores possíveis são **OFF**: Sinal **RTS** sempre desligado, **ON**: Sinal **RTS** sempre ligado ou **Toggle**: Liga (**ON**) o sinal **RTS** quando está transmitindo dados e desliga (**OFF**) o sinal **RTS** quando não está transmitindo dados.

IO.Serial.StopBits

Especifica o número de bits de parada para a configuração da porta serial. Os valores possíveis são **1**: Um bit de parada, **2**: Um bit e meio de parada ou **3**: Dois bits de parada.

IO.Serial.SupressEcho

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 um Driver a esperar pelo sinal **CTS** antes de enviar bytes quando o sinal **RTS** está ligado (**ON**). Disponível apenas quando a propriedade **IO.Serial.RTS** está configurada com o valor **Toggle**.

Histórico de Revisões do Driver

VERSÃO	DATA	AUTOR	COMENTÁRIOS
4.0.18	01/10/2026	C. Mello	• Adicionado suporte para que o usuário possa selecionar Tags PLC ou Bloco que não participam do agrupamento de leitura por **Superblocos** (*Case 40969*).
4.0.17	10/09/2026	M. Ludwig	• Adicionado suporte a comunicação segura em conformidade com a especificação **Modbus/TCP Security** (**TLS 1.2**, validação de certificados e autenticação mútua) (*Case 40632*).
4.0.16	28/07/2026	C. Mello	• Adicionado suporte para configuração do modo **Auto-Demotion** em tempo de execução (*Case 40345*).
4.0.15	04/05/2026	C. Mello	• Adicionado suporte para comunicar com equipamentos **Escravos** em modo **Auto-Demotion** (*Case 39490*).
4.0.14	16/03/2026	M. Ludwig	• Driver atualizado para a biblioteca **IOKit** versão **3.0** e Visual Studio 2022 (*Case 37995*).
4.0.13	09/01/2025	C. Mello	• Ajustes para prevenir travamentos no

VERSÃO	DATA	AUTOR	COMENTÁRIOS
			componente do tipo **List Control** na aba **Operations** da janela de propriedades deste Driver (*Case 37144*).
4.0.12	10/12/2024	M. Ludwig	• Adicionada, via biblioteca **IOKit**, uma configuração do tamanho máximo de arquivo de log na aba **Setup** da janela de propriedades (*Case 36834*).
4.0.11	30/08/2023	C. Mello	• Ajustes nos **Superblocos** para interpretação correta dos bits ao configurar Tags por **Strings**, a mesma usada na opção **Use Bit Mask** da configuração **Numérica** (*Case 34655*).
4.0.10	21/03/2023	C. Mello	• Adicionada a reconexão automática para o protocolo serial **Modbus RTU/ASC**, ao comunicar via **Ethernet TCP/IP** (*Case 16216*).
4.0.9	07/12/2022	C. Mello	• Ajustes para resgate de pacotes **Modbus RTU** válidos em encapsulamento TCP contendo bytes excedentes (*Case 33398*). • Corrigida uma inconsistência nos valores de tipos de dados **String** quando lidos via Superblocos (*Case 33313*).
4.0.8	27/09/2022	C. Mello	• Adicionado um tipo de dados Inteiro de 64 bits, convertido para o formato **String** (*Case 33213*).
4.0.7	01/12/2021	C. Mello	• Adicionado suporte a Superblocos em Tags configurados por **String** (*Case 20900*).
4.0.6	07/04/2021	C. Mello	• Adicionado um tratamento de qualidade para valores com o tipo de dados **Float** não normalizados (*Case 30742*).
4.0.5	05/11/2020	C. Mello	• Adicionado suporte para endereçamento de **Escravos** com um identificador igual a 0 (zero) no modo **Modbus TCP** (*Case 29886*).

VERSÃO	DATA	AUTOR	COMENTÁRIOS
4.0.4	22/06/2020	C. Mello	• Adicionado um tipo de dados Inteiro de 52 bits via conversão para um tipo de dados **Double** de 64 bits (*Case 21840*).
4.0.3	17/03/2020	C. Mello	• Corrigido um problema de configuração *offline* para as **Strings** *ModiconModbus.ConfigFile* e *ModiconModbus.UserTypesConfigFile* (*Case 28566*).
4.0.2	05/08/2019	M. Ludwig	• Driver portado para o Visual Studio 2017 (*Case 27087*).
4.0.1	13/11/2018	C. Mello	• Corrigido um problema com a configuração **1000**, que vincula um Tag com o **Default Slave Address** da janela de propriedades (*Case 20713*). • Implementado o retorno para tipos de dados **BYTE** para os valores de registradores Coils (*Case 20927*). • Corrigido um problema com as escritas de tipos de dados **String** com tamanho ímpar e com a opção **Swap bytes / words / dwords** habilitada (*Case 23694*). • Ajustes de consistência nas edições de estruturas de tipos de dados definidos pelo usuário (*Case 20454*).
3.1.36	17/03/2017	C. Mello	• Adicionado suporte para inteiros com sinal de magnitude (*Case 22091*).
3.1.31	10/07/2016	F. Englert	• Resolvido um GPF (*General Protection Failure*) que poderia ocorrer eventualmente na leitura do histórico de erros deste Driver, com o parâmetro *B2* igual a 9998, caso ocorresse ao mesmo tempo algum erro de comunicação (*Case 21637*).
3.1.30	01/04/2016	C. Mello	• Corrigida a falta de valores dos Tags configurados por **Strings** quando o serviço

VERSÃO	DATA	AUTOR	COMENTÁRIOS
			de Superblocos está ativado (*Case 20754*).
3.1.28	28/01/2016	A. Quites	• Implementado um novo Tag Browser com *templates* de Tags configurados por **Strings** (*Case 20460*).
3.1.27	27/01/2016	A. Quites	• Implementada uma opção de endereçamento de Tags por **Strings** nos parâmetros **Item** e **Device** (*Case 19119*). • Corrigido um erro na leitura de Tags de tempo real de eventos de equipamentos GE PAC RX7 por *callbacks* (*Case 20374*).
3.1.26	20/01/2016	A. Quites	• Corrigido um erro em que a rotina **Elipse SOE** não retornava valores reportados a eventos para tipos de dados nativos, não estruturados (*Case 20364*).
3.1.25	19/01/2016	A. Quites	• Implementada a configuração por **Strings** para tipos de dados do usuário (*Case 19807*).
3.1.24	18/01/2016	A. Quites	• Adicionada uma verificação para proibir a definição de tipos de dados do usuário com nomes que conflitem com tipos de dados nativos deste Driver (*Case 19816*). • Corrigido um erro na leitura de tipos de dados **UTC32** em Bloco, que retorna valores errados a partir do segundo Elemento de Bloco (*Case 19819*).
3.1.23	14/01/2016	A. Quites	• 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 (*Case 20415*).
3.1.15	22/12/2015	A. Quites	• Corrigida a escrita em bloco de Tags Bloco com tipo de dados **Double** (*Case 20053*).
3.1.13	02/12/2015	A. Quites	• Corrigido um erro na exportação de arquivos INI em formato compatível com a versão **1.0** deste Driver,

VERSÃO	DATA	AUTOR	COMENTÁRIOS
			com os tipos de dados especificados por números e não por **Strings**, como passou a ocorrer a partir da versão **2.8** (*Case 20203*). • Corrigido um erro de *byte order* na escrita de tipos de dados **BCD** de 8 (oito) dígitos e tamanho de 4 (quatro) bytes (*Case 20204*).
3.1.9	01/10/2015	A. Quites	• Habilitado o armazenamento de últimas exceções para Tags de comunicação configurados por **Strings** (*Case 19808*).
3.1.8	28/09/2016	A. Quites	• Corrigido um erro na leitura e escrita de tipos de dados **BCD** de 8 (oito) dígitos e 4 (quatro) bytes superiores a 9999999, que provocavam a finalização do processo (*Case 19733*).
3.0.11	29/05/2015	A. Quites	• Driver alterado para evitar possível comportamento estranho na reconexão após *time-out* em Tags de *callback* (SOE), com a opção **Reconnect after Timeout (Ethernet only)** habilitada. Podem ocorrer, por exemplo, 2 (duas) conexões simultâneas seguidas de 2 (duas) desconexões (*Case 14775*). • Corrigido um erro na gravação de operações na janela de configuração deste Driver, na aba **Operações**, que fazia com que, ao remover operações no final da lista, eventualmente voltassem à lista quando a janela era fechada clicando em **OK** e reaberta logo em seguida (*Case 14874*). • Corrigido um problema que poderia gerar a perda de informações de última exceção em casos raros em que são recebidos *frames* válidos com CRC correto, porém que não se

VERSÃO	DATA	AUTOR	COMENTÁRIOS
			destinavam à requisição atual. Pode ocorrer quando se utiliza **Modbus RTU** em meio **Ethernet TCP/IP** em redes lentas (*Case 15314*). • Solucionado um vazamento de *handle* do *thread* de *download* de memória de massa de um equipamento GE PAC RX7 (*Case 16404*). • Corrigido um erro em que este Driver poderia ignorar os últimos caracteres de **Strings** lidas de um equipamento quando alguma das opções de *swap* estavam configuradas na operação (*Case 16744*). • Corrigido um erro na leitura de operações com a opção **Use bit mask** habilitada quando usada com a propriedade **EnableReadGrouping** deste Driver habilitada (*Case 18340*). • Resolvido um erro de validação em Tags especiais deste Driver (*Case 16433*).
3.0.8	31/07/2014	A. Quites	• Corrigido um erro que poderia gerar um GPF (*General Protection Failure*) ou comportamentos inesperados ao executar múltiplas instâncias deste Driver em um mesmo I/O Server, sobretudo se estas instâncias possuíssem configurações diferentes nas abas **Modbus** e **Operations** da janela de configurações deste Driver (*Case 14856*).
3.0.6	12/06/2014	A. Quites	• Driver portado para a biblioteca **IOKit** versão **2.0** (*Case 13891*).
2.8.17	19/10/2012	A. Quites	• Adicionados os tipos de dados ou estruturas definidas pelo usuário como parte da implementação do recurso de SOE Genérico (*Case 12038*).

VERSÃO	DATA	AUTOR	COMENTÁRIOS
			• Implementada a leitura de SOE e comandos para a sincronização do relógio para relés Schneider Electric das séries SEPAM 20, 40 e 80 (*Case 12106*). • A opção **Reverse Frame** foi removida da caixa de diálogo de configuração das operações deste Driver, por obsolescência. A opção continua suportada em aplicações legadas somente como configuração *offline* (*Case 12443*). • Adicionada a leitura por *callbacks* para Tags de leitura de SOE (*Case 12464*). • Adicionada uma opção de reconexão em caso de ocorrência de *time-out* na recepção de *frames* ao usar a camada física Ethernet (*Case 12537*). • A opção **Swap Address Delay** foi removida da janela de configuração deste Driver. Esta opção, tornada obsoleta pela opção **Inter-frame delay** da biblioteca **IOKit**, continua disponível como configuração *offline*, mantendo compatibilidade com aplicações legadas (*Case 13285*). • Solucionado um erro na escrita de valores **float_GE** (*Case 12298*). • Solucionado um erro ao carregar um arquivo de configuração no Windows CE ARM HPC2000 (*Case 12352*). • Solucionado um erro em que a leitura de **Strings** de tamanho ímpar poderia vir sem o último caractere (*Case 12466*).
2.7.1	30/06/2010	A. Quites	• Implementado o recurso de tornar este Driver imune a ruídos antes de receber o

VERSÃO	DATA	AUTOR	COMENTÁRIOS
			frame em modo **RTU** (*Case 11394*).
2.6.1	26/11/2009	A. Quites	• Driver portado para o **Windows CE** (*Case 10914*). • Driver reporta um falso erro no log ao ler um endereço de registro em zero (*Case 10654*). • Otimizada a leitura de bits usando o agrupamento de leituras ou Superblocos (*Case 10971*).
		C. Mello	• A opção **Wait Silence** não funciona para todos os erros (*Case 10850*).
2.5.1	30/06/2009	A. Quites	• Atualizado o tamanho máximo do PDU para este Driver (*Case 10274*). • Corrigido um erro ao ler a função Modbus **20**, **Read File Record** (*Case 10312*). • Corrigido um erro no Tag Especial para retorno da última exceção, que poderia não reportar algumas exceções (*Case 10337*). • Eventos **GE SOE** com o dia atual retornavam o ano errado (*Case 10382*). • Endereço de *swap* não funcionava (*Case 10425*).
		M. Ludwig	• Criado um novo Tag para receber uma lista de eventos de pontos específicos no **GE SOE** (*Case 10370*).
2.4.1	17/02/2009	A. Quites	• Adicionadas funções de leitura e escrita por padrão (*Case 9185*). • Driver com Eventos **GE**, leitura de último evento por *polling* (*Case 10178*). • Erro ao ler blocos de bits com agrupamentos de leituras ou Superblocos habilitados (*Case 10100*).
2.3.1	02/09/2008	A. Quites	• Implementado o recurso **CMS Extended Device Addressing** (*Case 8665*).

VERSÃO	DATA	AUTOR	COMENTÁRIOS
			• Implementada a configuração de tamanho máximo dos Superblocos (*Case 9154*). • Este Driver aceita enviar um **DWord** de 32 bits usando a função Modbus **6** (*Case 8663*). • Revisão das funções não padrão ou pouco utilizadas **07**, **20**, **21**, **65 01**, **65 02** e **65 03** (*Case 8730*). • A opção **Swap byte** não funciona corretamente com Superblocos (*Case 9220*). • Configuração em modo *offline* do parâmetro *ModiconModbus.ModbusMode* (*Case 9831*).
2.2.1	11/05/2007	A. Quites	• Usar o tipo de dados **Int16** para a leitura de blocos retorna somente o primeiro elemento (*Case 8243*).
2.1.1	23/01/2007	A. Quites	• Implementados os Superblocos (*Case 6185*). • O **Mestre** Modicon Modbus não testa corretamente a consistência do parâmetro *N2* ou *B2* (*Case 7714*). • Erro de *offset* ao ler blocos de tipos de dados **BCD** com o tamanho do tipo de dados igual a quatro (*Case 7728*). • Leitura de blocos de **Strings** não estava funcionando corretamente (*Case 7804*).
2.0.1	14/09/2005	A. Quites	• Driver portado para a biblioteca **IOKit** (*Case 2050*).
1.3.1	19/12/2006	C. Mello	• Driver comunicando com mais de um endereço IP muda os valores lidos (*Case 7191*). • Possibilidade de definir uma porta TCP/IP local (*Case 7109*). • Corrigido um erro de *offset* ao ler blocos de tipos de dados **BCD** com o tamanho

VERSÃO	DATA	AUTOR	COMENTÁRIOS
			do tipo de dados igual a quatro (*Case 7729*). • Driver não testava a consistência dos parâmetros *N2* ou *B2* corretamente (*Case 7735*).
1.2.1	15/12/2005	C. Mello	• Ajustes para comunicação com Corretores de Vazão da ICP (*Case 4979*). • Corrigido um vazamento de *handles* se o comando **ping** falhasse (*Case 6497*).
1.1.1	22/12/2004	C. Mello	• Adicionadas retentativas a este Driver (*Case 3365*). • Adicionada a função **Broadcast** (*Case 4045*). • Adicionada a opção **HALT** a este Driver (*Case 4429*). • Corrigido um erro na leitura e escrita de **Strings** (*Case 4386*). • Corrigidas as configurações incompatíveis com versões anteriores (*Case 4431*).

Matriz
Rua Mostardeiro, 322/Cj. 902, 1001 e 1002
90430-000 — Porto Alegre — RS
Fone: (+55 51) 3346-4699
Fax: (+55 51) 3222-6226
E-mail: elipse-rs@elipse.com.br

Filial no Paraná
Av. Sete de Setembro, 4698/1708
80240-000 — Curitiba — PR
Fone: (+55 41) 4062-5824
E-mail: elipse-pr@elipse.com.br

Filial no Rio de Janeiro
Av. José Silva de A. Neto, 200/Bl. 4/Sl. 109B
22250-044 — Rio de Janeiro — RJ
Fone: (+55 21) 2430-5912
Suporte Técnico: (+55 21) 2430-5963
E-mail: elipse-rj@elipse.com.br

Filial em São Paulo
Rua dos Pinheiros, 870/Cj. 141 e 142
05422-001 — São Paulo — SP
Fone: (+55 11) 3061-2828
Fax:(+55 11) 3086-2338
E-mail: elipse-sp@elipse.com.br

Filial em Minas Gerais
Rua Antônio de Albuquerque, 156/705
30112-010 — Belo Horizonte — MG
Fone: (+55 31) 4062-5824
E-mail: elipse-mg@elipse.com.br

Filial em 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

Consulte nosso website para informações sobre o representante do seu estado.

www.elipse.com.br
kb.elipse.com.br
forum.elipse.com.br
www.youtube.com/elipsesoftware
elipse@elipse.com.br

Gartner 2014 Cool Vendor

OPC FOUNDATION MEMBER

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)

Copyright © 2001-2026 Elipse Software Ltda. Todos os direitos reservados.