

Driver Modicon Modbus Master (ASC/RTU/TCP)



Nome do Arquivo	MODBUS.DLL
Fabricante	Modicon
Equipamentos	Qualquer equipamento compatível com o protocolo Modbus v1.1b
Protocolo	Modbus v1.1b
Versão	3.1.29
Última Atualização	22/02/2016
Plataforma	Win32 e Windows CE (Pocket PC ARM, HPC2000 ARM, HPC2000 X86 e HPC2000 MIPS)
Dependências	Sem dependências
Leitura com Superblocos	Sim
Nível	0

Introdução

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

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

O Driver Modbus, a partir da versão 2.00, passou a ser desenvolvido com a biblioteca **IOLib** da Elipse. Esta biblioteca é responsável por implementar o acesso ao meio físico (**Serial**, **Ethernet**, **Modem** ou **RAS**). Para obter informações sobre a configuração do IOLib, consulte o **Manual do Usuário do IOLib**.

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

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

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

Caso não conheça o protocolo, consulte os seguintes tópicos:

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

Guia de Configuração Rápida

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

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

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

Inserindo o Driver

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

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

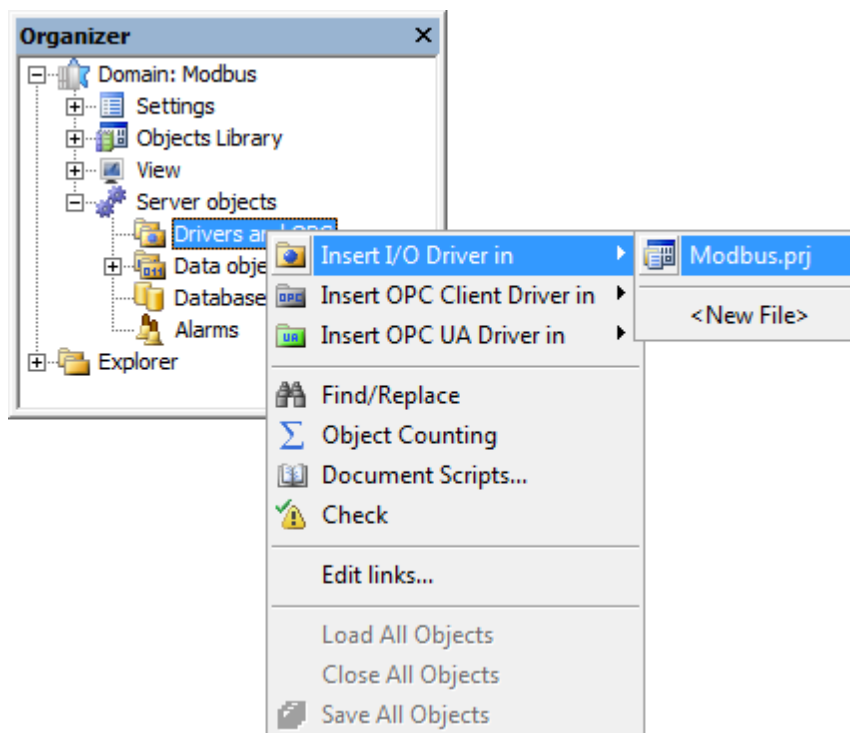
Depois, leia o **passo seguinte** deste Manual, que ensina como configurar o Driver em sua janela de configuração para os casos mais comuns.

Adicionando o Driver em uma Aplicação da Elipse Software

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

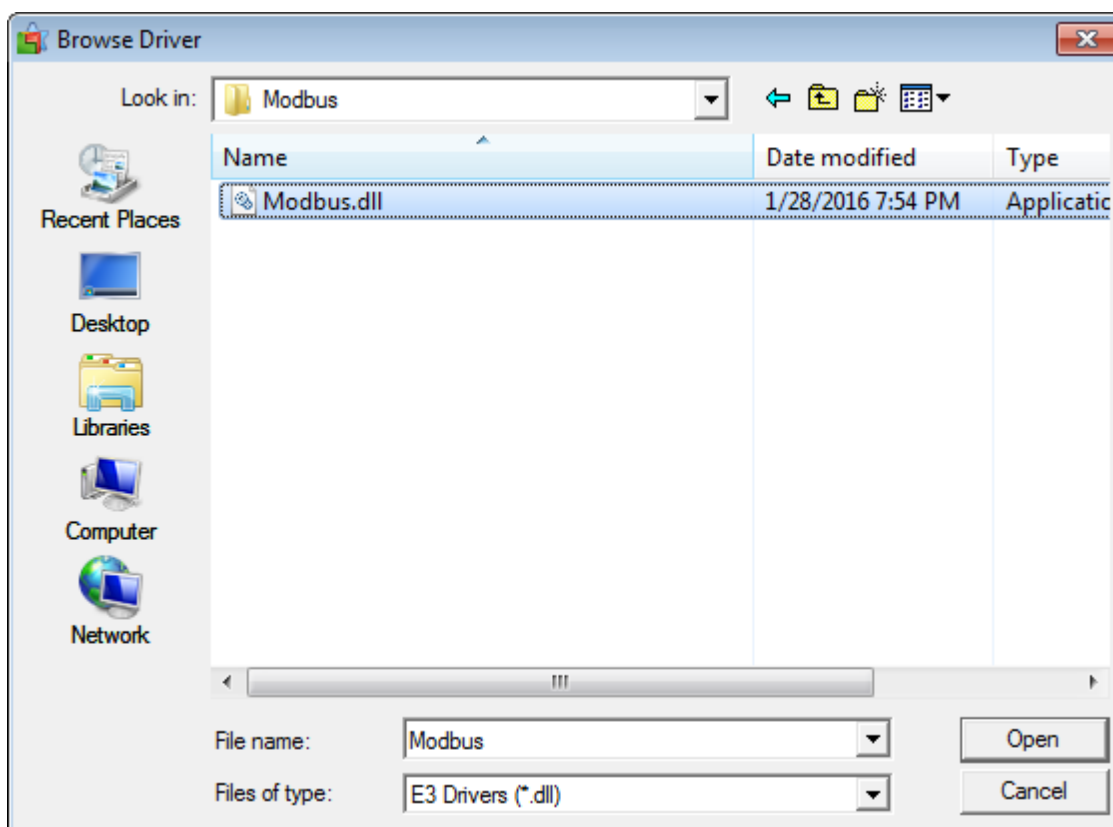
E3 ou Elipse Power

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



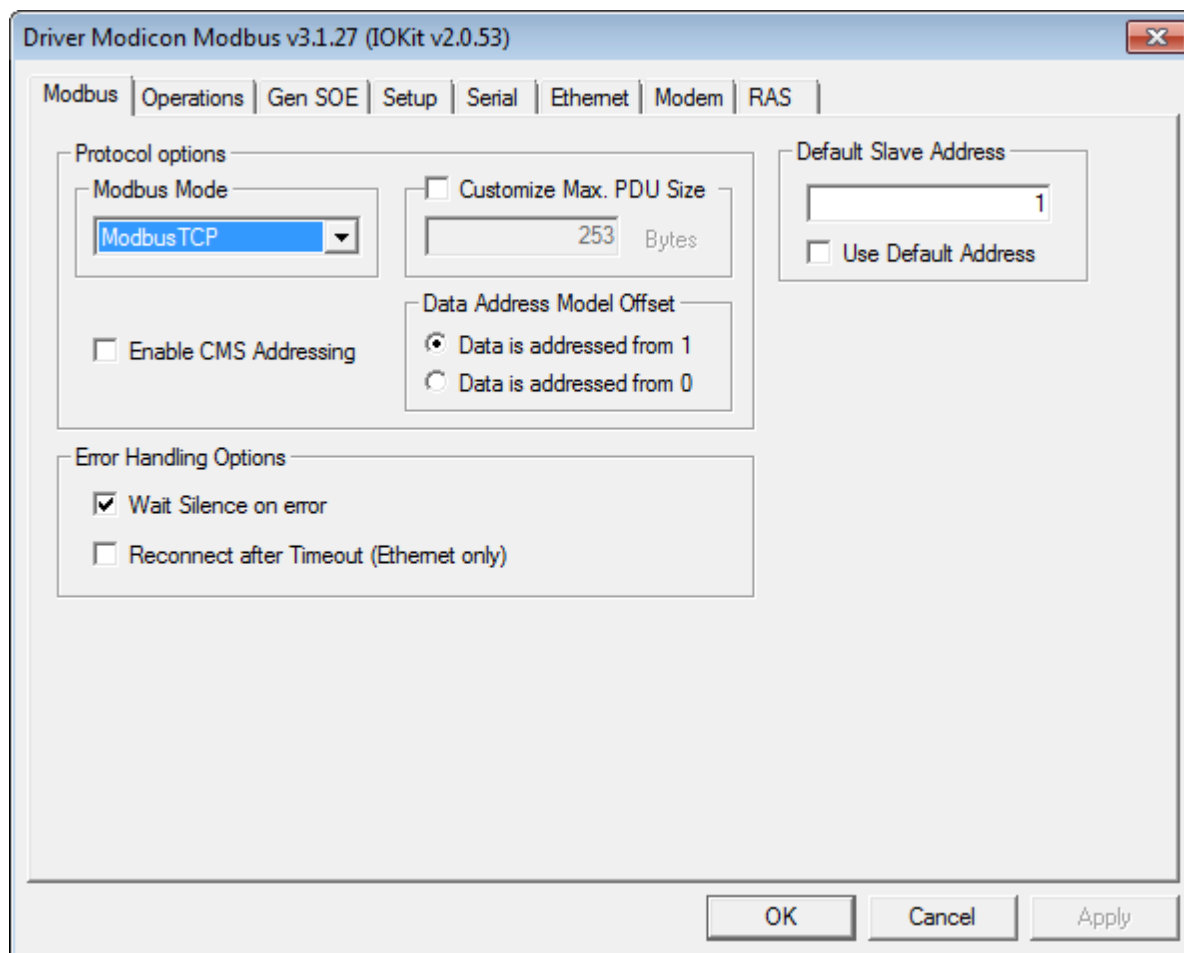
Adicionar um Driver a uma aplicação E3 ou Elipse Power

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




Selecionando o Driver

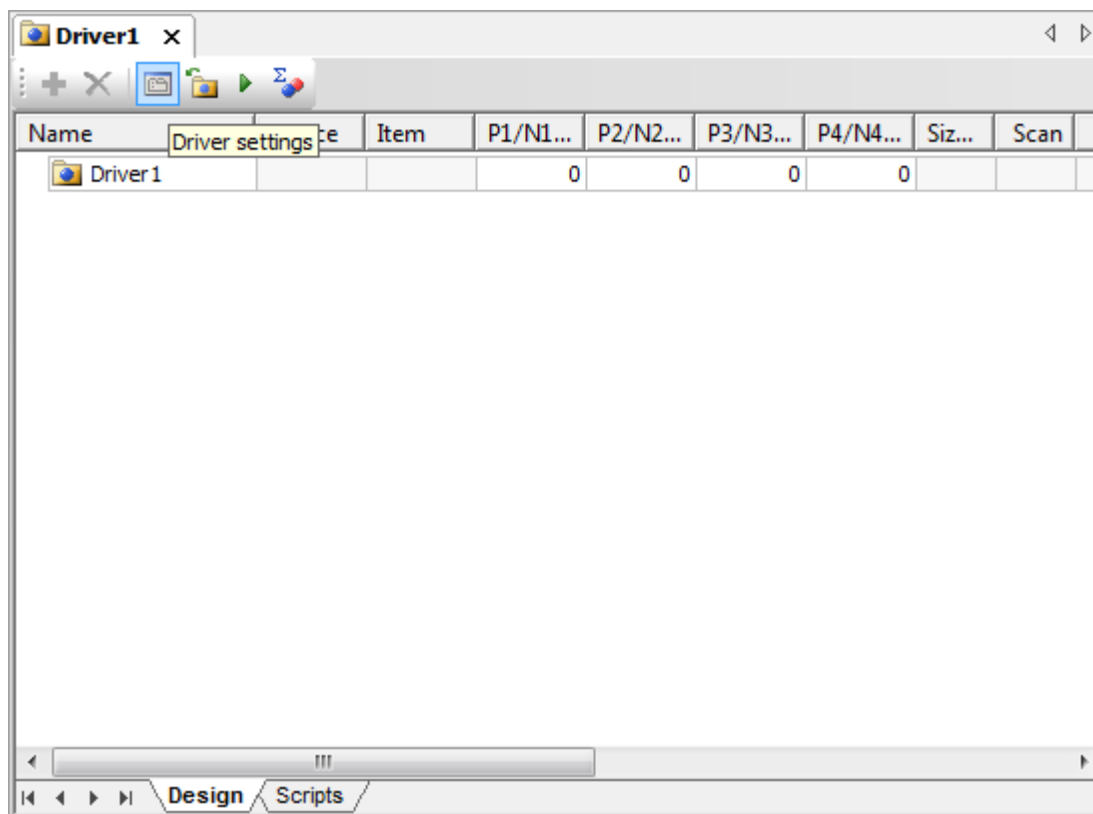
A janela de configurações do Driver abre-se automaticamente, conforme mostrado na figura a seguir.



Janela de configurações do driver

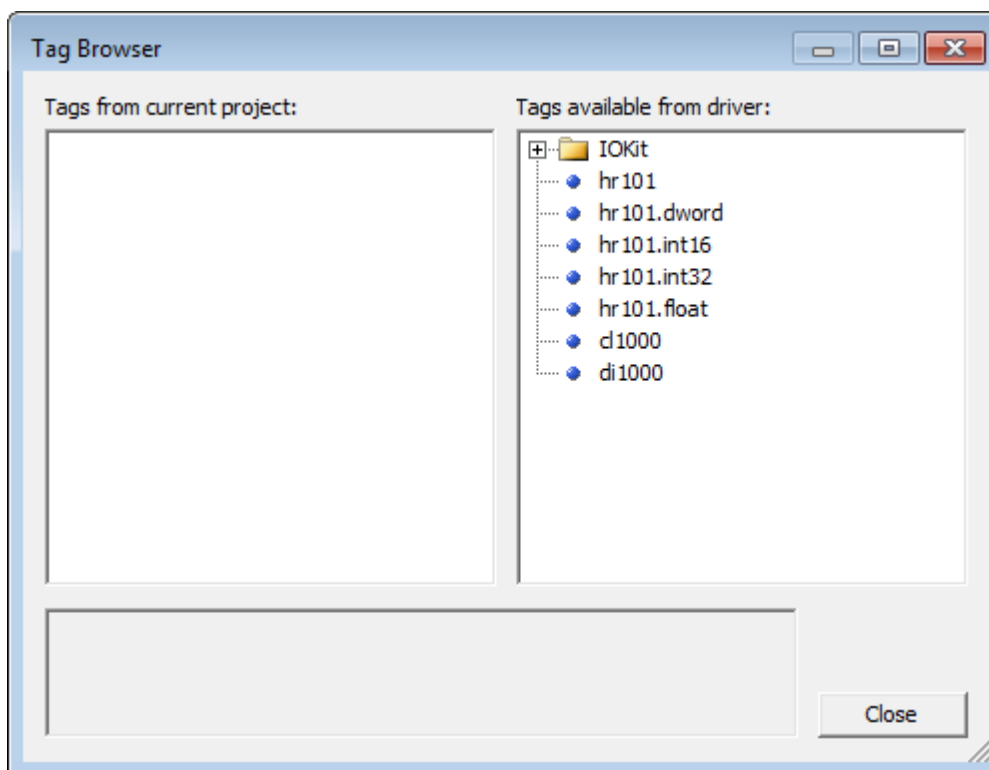
No **segundo passo** do tópico **Guia de Configuração Rápida** é apresentado o procedimento para a configuração básica do Driver, para os casos de uso mais comuns. No tópico **Propriedades** a configuração é apresentada em detalhes.

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




Opção Driver settings (Configurar o driver)

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

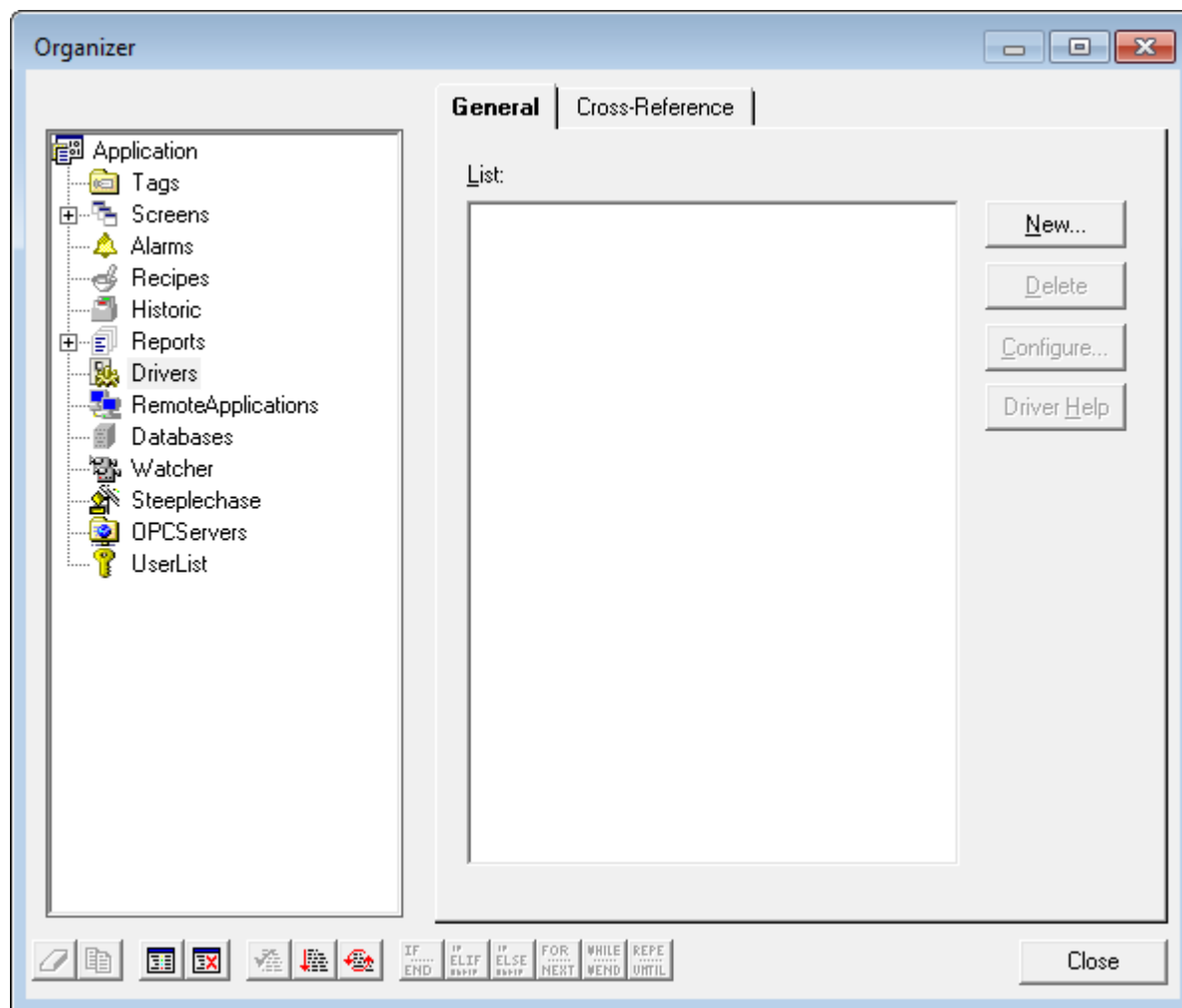


Janela Tag Browser

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

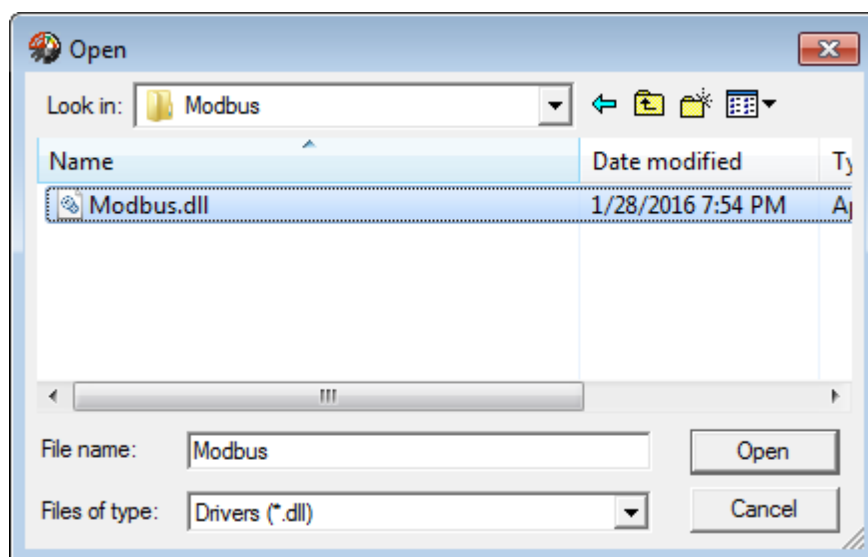
Elipse SCADA

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



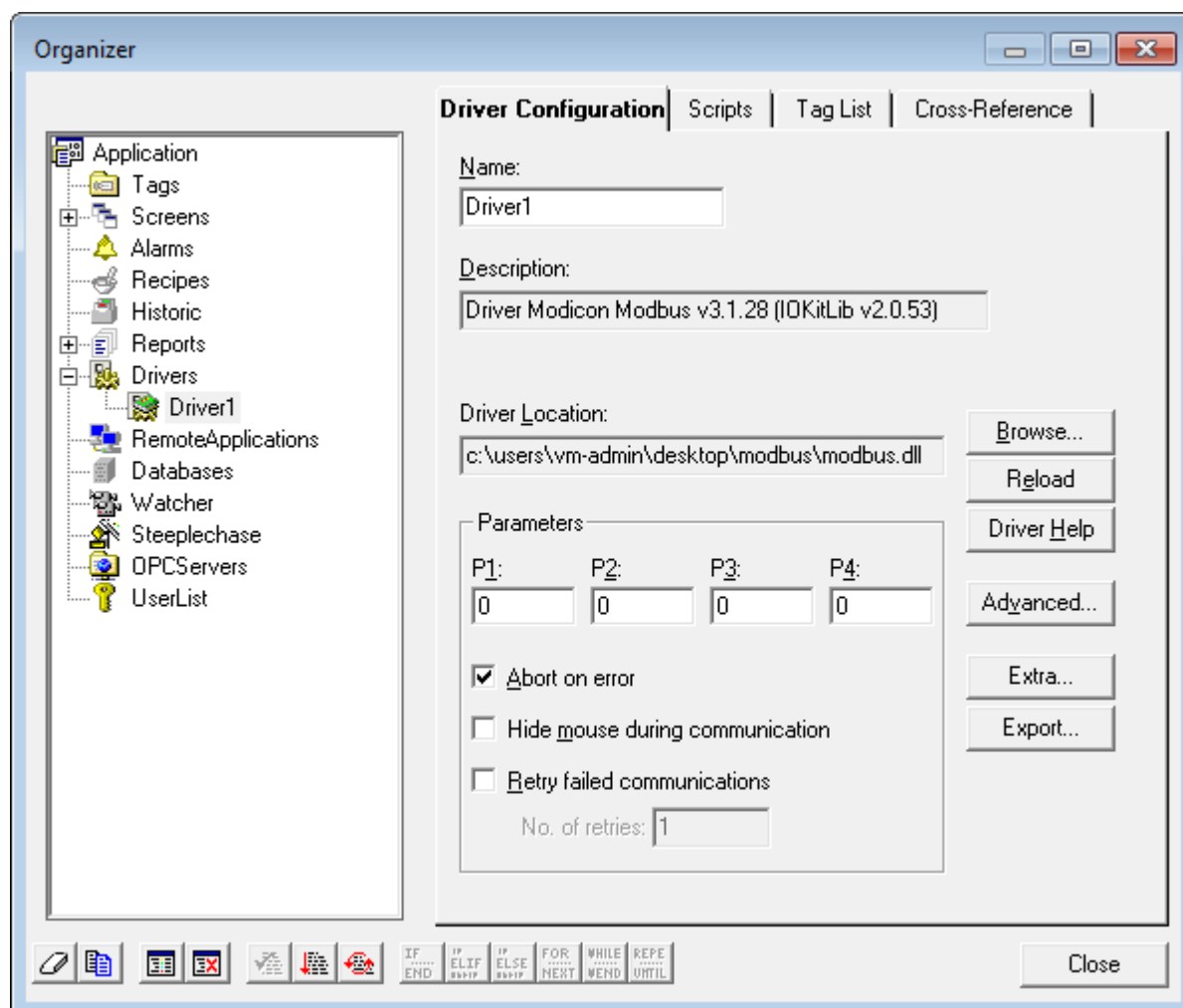
Adicionar novo Driver ao Elipse SCADA

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



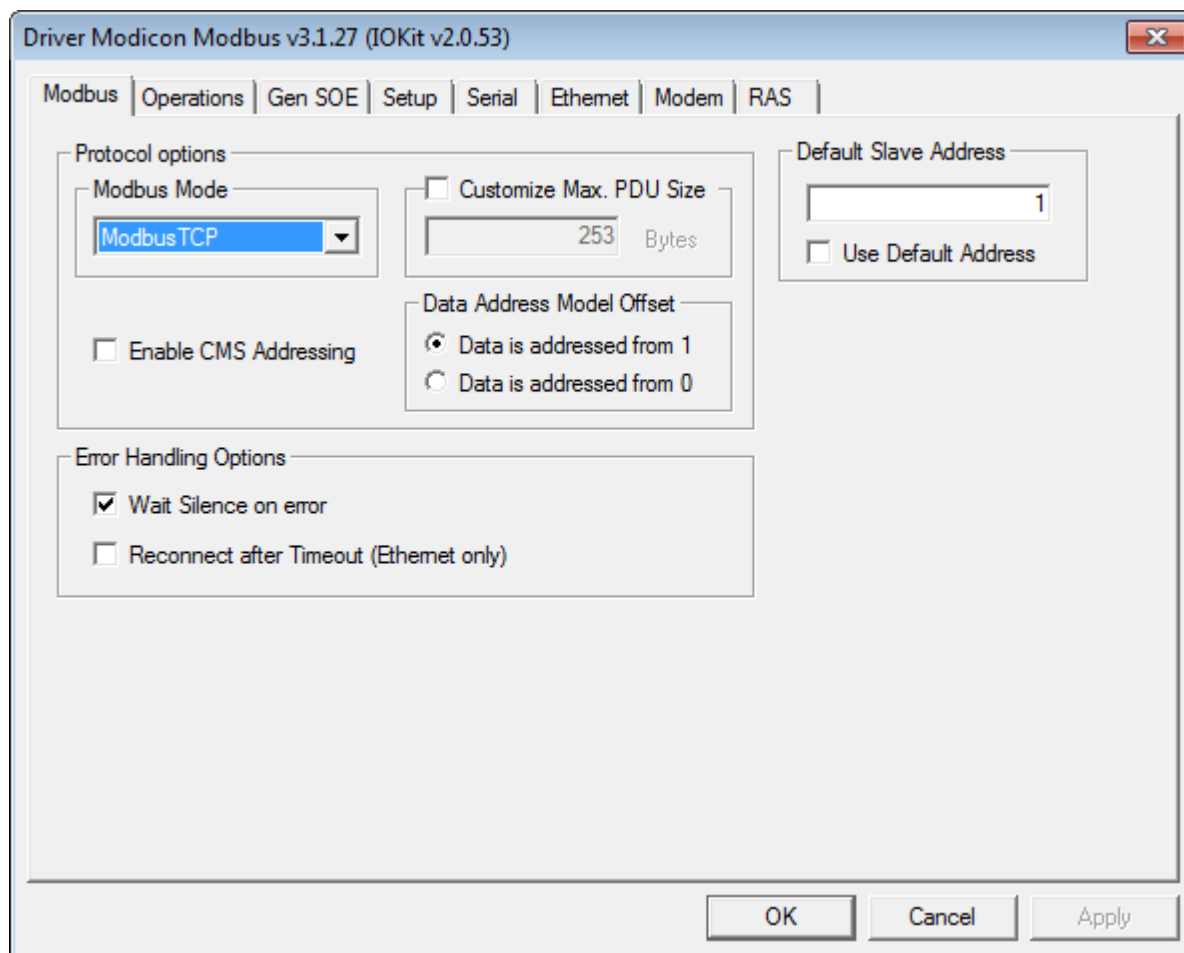
Selecionando Driver

O Driver é então adicionado na aplicação.



Driver no Organizer

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



Janela de configuração do Driver

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

Configurando o Driver

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

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

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

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

Operações pré-definidas

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

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

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

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

Configurando Tags de Comunicação


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

Configuração de Tags no E3 e no Elipse Power

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

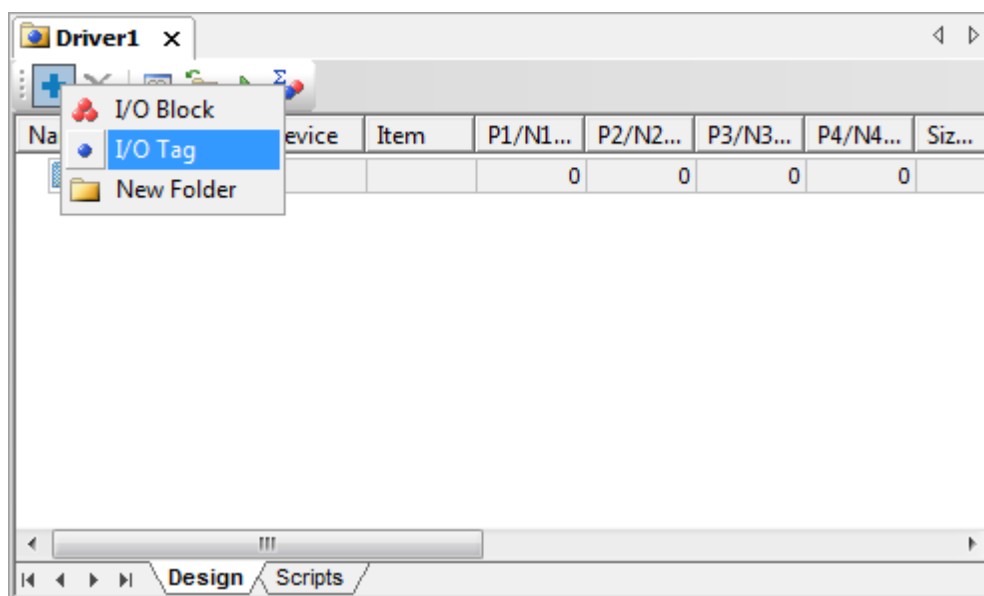
A seguir estão descritos os procedimentos recomendados para a configuração por **Strings** e também para a antiga configuração numérica, caso seja necessária para aplicações legadas.

Configuração por Strings

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

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

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



Adicionar novo Tag de Comunicação

2. Na janela **Adding IOTag (Inserindo IOTag)**, configure o campo **Quantity (Quantity)** com o valor 1 (um) e especifique um nome para o Tag no campo **Name (Nome)**. Clique em **OK** para criar o novo Tag.
3. Na coluna **Device (Dispositivo)**, digite o valor numérico do *Slave Id* do equipamento a comunicar, seguido de dois pontos, como por exemplo "1:" para um *Slave Id* igual a 1 (um). Note que, em um meio **Ethernet TCP/IP**, muitas vezes este valor é ignorado, sendo utilizado apenas o endereço IP e a porta configurada na aba **Ethernet**, que deve constar na documentação do equipamento.

4. Na coluna **Item**, especifique o mnemônico do **espaço de endereçamento** (conjunto de funções Modbus de leitura e escrita) seguido do endereço do registro ou bit. Para *Holding Registers*, o espaço de endereçamento é "hr" ou "shr", onde este último não permite escrita em blocos, pois usa a função de escrita **06** (*Write Single Register*), enquanto o espaço de endereçamento "hr" usa a função de escrita **16** (*Write Multiple Registers*). Ambos usam a função Modbus de leitura **03** (*Read Holding Registers*). Para *Coils* utilize "cl" ou "scl". Novamente, a diferença é que este último, que usa a função **05** (*Force Single Coil*), não escreve em blocos. A seguir são fornecidos alguns exemplos de configuração da coluna **Item**.
- a. Leitura ou escrita do *Holding Register* 150, usando as funções **03** e **16** (escrita de múltiplos registros): **Item** deve ser igual a "hr150".
 - b. Leitura ou escrita do *Holding Register* 150, usando as funções **03** e **06** (escrita de registros simples): **Item** deve ser igual a "shr150".
 - c. Leitura ou escrita do *Coil* de endereço FFF0h (65520), usando as funções **01** e **15** (escrita de múltiplos bits): **Item** deve ser igual a "cl65520" ou "cl&hFFF0" (o prefixo "&h" pode ser usado para fornecer endereços em formato hexadecimal).
 - d. Leitura ou escrita do *Coil* de endereço FFF0h (65520) usando as funções **01** e **05** (escrita de bits simples, um a um): **Item** deve ser igual a "scl65520" ou "scl&hFFF0" (o prefixo "&h" pode ser usado para fornecer endereços em formato hexadecimal).
5. Para mais informações sobre os demais recursos da configuração por **Strings**, como outras funções Modbus, funções especiais e diferentes tipos de dados, consulte o tópico **Configuração por Strings**.
6. O endereçamento dos Tags deve corresponder ao mapa de endereços Modbus do equipamento, que deve constar na documentação do fabricante. Em caso de dúvida, consulte o tópico **Dicas de Endereçamento**.

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

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

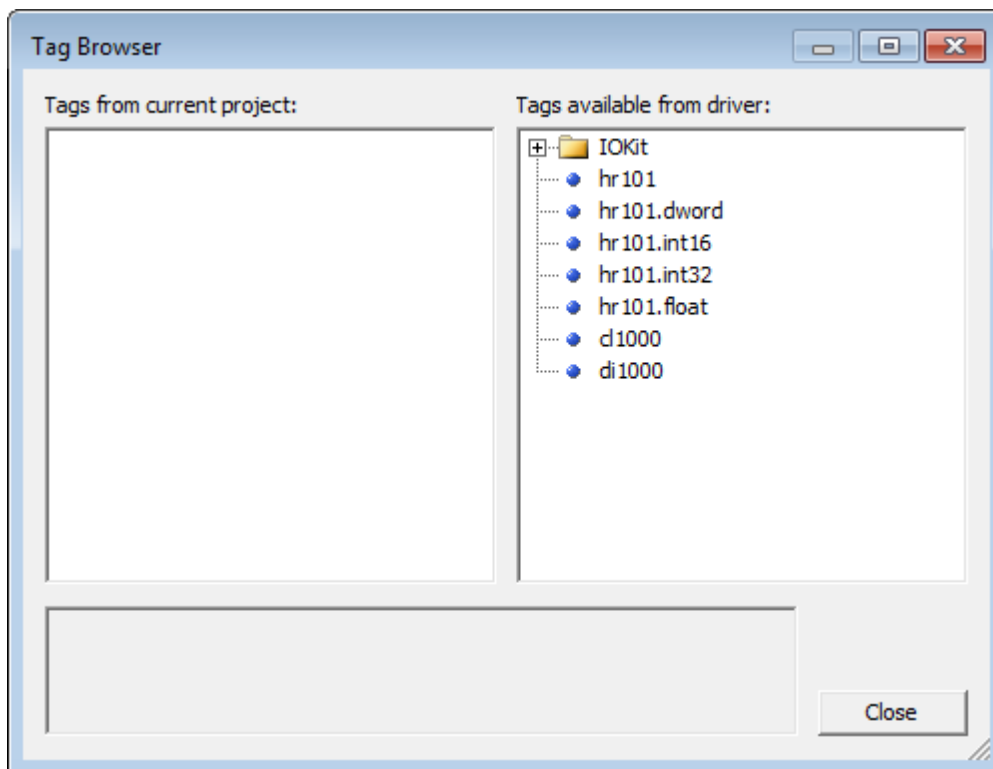
Name	Device	Item	P1/N1...	P2/N2...	P3/N3...	P4/N4...
Driver1			0	0	0	0
Analog						
Temperature_1	1:	hr1	0	0	0	0
Temperature_2	1:	hr2	0	0	0	0
Level_T1	1:	hr3	0	0	0	0
Level_T2	1:	hr4	0	0	0	0
Digital						
Status_B1	1:	sd1	0	0	0	0
Status_B2	1:	sd2	0	0	0	0
Status_B3	1:	sd3	0	0	0	0
Status_B4	1:	sd4	0	0	0	0
Status_B5	1:	sd5	0	0	0	0
Status_B6	1:	sd6	0	0	0	0
AutoMan_B1	1:	sd7	0	0	0	0
AutoMan_B2	1:	sd8	0	0	0	0
AutoMan_B3	1:	sd9	0	0	0	0
AutoMan_B4	1:	sd10	0	0	0	0
AutoMan_B5	1:	sd11	0	0	0	0
AutoMan_B6	1:	sd12	0	0	0	0
Failure_B1	1:	sd13	0	0	0	0
Failure_B2	1:	sd14	0	0	0	0
Failure_B3	1:	sd15	0	0	0	0
Failure_B4	1:	sd16	0	0	0	0
Failure_B5	1:	sd17	0	0	0	0
Failure_B6	1:	sd18	0	0	0	0

Exemplo de Tags configurados por Strings

Configuração Numérica

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

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



Tag Browser para a configuração numérica de Tags

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

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

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

Configuração de Tags no Elipse SCADA

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

- **N1/B1**: Endereço do Equipamento (*Slave Id*)

- **N2/B2:** Código da operação
- **N3/B3:** Não usado, deixar em 0 (zero)
- **N4/B4:** Endereço do registro Modbus ou bit

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

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

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

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

Considerações Finais

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

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

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

O Protocolo Modbus

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

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

- **Códigos de funções públicas:** Funções bem definidas pelo protocolo, com garantia de unicidade, validadas pela comunidade Modbus e publicamente documentadas em MB IETF RFC. Podem assumir valores de **1 a 64**, de **73 a 99** e de **111 a 127**.
- **Códigos de funções definidas pelo usuário:** Funções não padronizadas, que não precisam de aprovação da Modbus.org, sem qualquer garantia de unicidade, podendo ser livremente implementadas. Podem assumir valores nas faixas de **65 a 72** e de **100 a 110**.
- **Códigos de funções reservadas:** Códigos com valores dentro da faixa de funções públicas, atualmente

usados por alguns fabricantes em produtos antigos, e não mais disponíveis para uso público. São exemplos os códigos **9, 10, 13, 14, 41, 42, 90, 91, 125, 126 e 127**. Para mais informações, consulte o **Anexo A** da especificação do protocolo (versão 1.1b), que está disponível no *site oficial do protocolo*.

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

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

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

Sites Recomendados

O Driver Modicon Modbus Master (ASC/RTU/TCP) está disponível para *download* (sem custos) no site da Elipse Software, na área de *download de Drivers*.

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

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

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

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

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

Funções Suportadas

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

Funções de Leitura

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

Funções de Escrita

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

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

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

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

Funções Especiais

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

Funções de Leitura

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

Funções de Escrita

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

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

Configuração

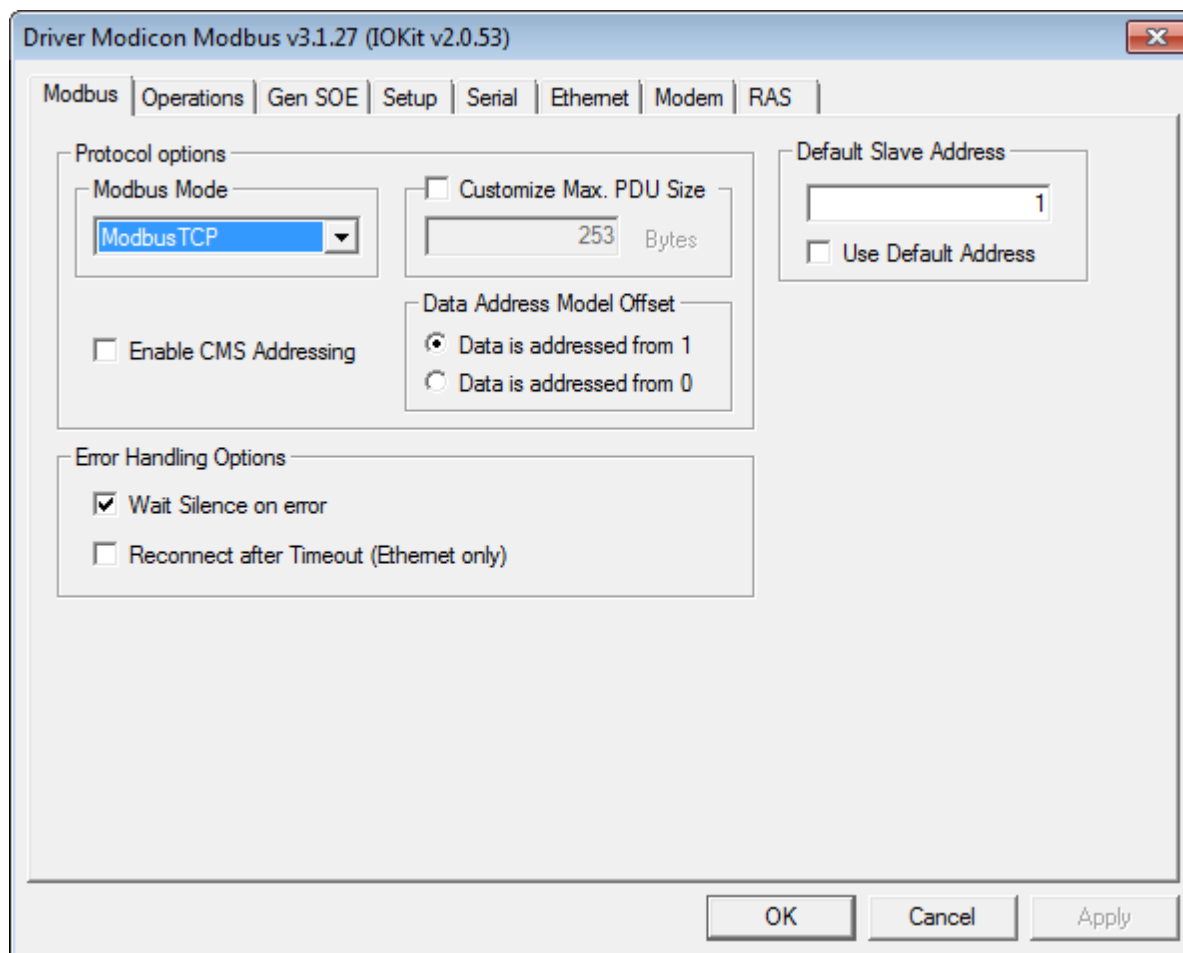
Esta seção descreve como configurar o Driver Modbus. São abordados os seguintes tópicos:

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


Propriedades

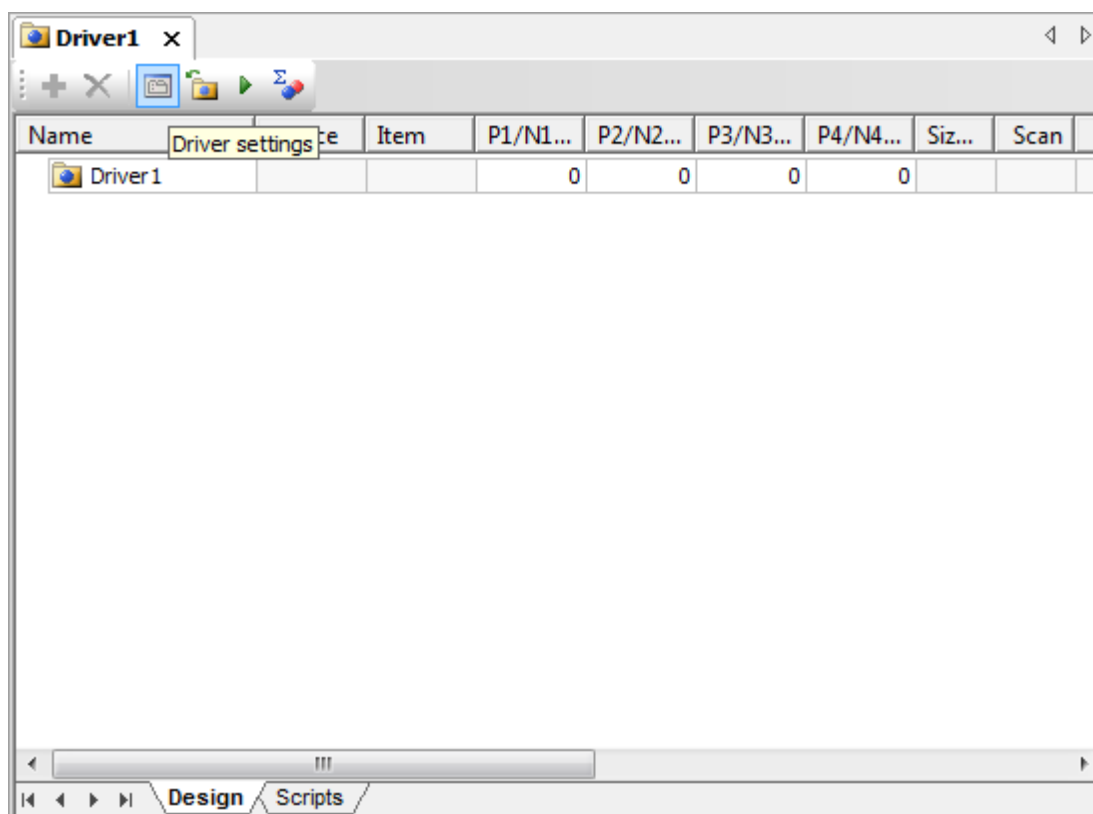
As propriedades do Driver podem ser configuradas em tempo de configuração (*design time*) ou em tempo de execução (*run time*). A configuração em tempo de execução (*run time*) também é chamada de **Configuração em Modo Offline** e é descrita em um tópico específico.

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



Janela de configuração do Driver

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



Opção Configurar o driver

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

da aplicação.

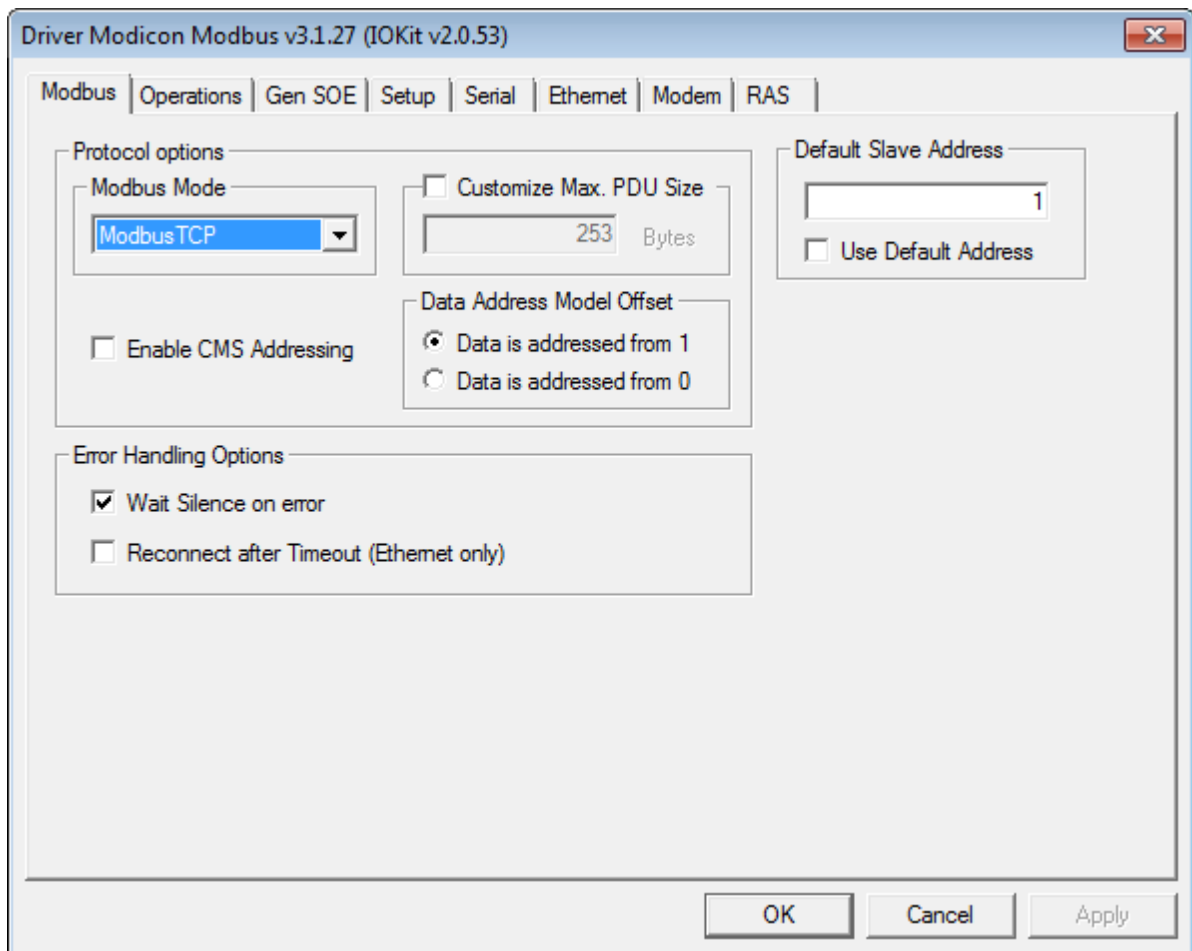
A janela de configuração é dividida em várias abas, algumas para a configuração do IOKit e outras específicas do Driver. No caso do Driver Modbus, as abas **Modbus**, **Operations** e **Gen SOE** são específicas. As demais abas são para configuração do IOKit e não são descritas neste Manual. Para mais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

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

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

Aba Modbus

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



Aba Modbus da janela de configurações do Driver

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

Protocol Options

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

seguir.

Opções de protocolo disponíveis na aba Modbus

OPÇÃO	DESCRIÇÃO
Modbus Mode	<p>Nessa caixa de seleção é possível selecionar o modo do protocolo a ser utilizado. Os modos do protocolo são variações definidas pela norma para melhor adaptá-lo a diferentes meios físicos (Serial, Ethernet TCP/IP, RAS, etc.). São três as opções disponíveis:</p> <ul style="list-style-type: none">• Modo RTU: Modo padrão para uso em comunicação serial. 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 dois caracteres ASCII (um por <i>nibble</i>), e por isto menos eficiente que o modo RTU e mais raramente encontrado no mercado. Usa LRC (<i>Longitudinal Redundancy Checking</i>) para a verificação de erros.• 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 o Driver assuma como resposta válida para o comando atual os <i>frames</i> de resposta a comandos anteriores. Esta situação pode ocorrer se os modos anteriores forem encapsulados em TCP/IP.
Customize Max. PDU Size	<p>Se habilitada, esta opção permite definir um tamanho máximo personalizado para o PDU (<i>Protocol Data Unit</i>). O PDU é a parte do protocolo que não varia entre os modos (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.</p>
Enable CMS Addressing	<p>Esta opção deve ser usada apenas em equipamentos que suportem o protocolo TeleBUS. Se habilitada, o Driver passa a aceitar um Word de 16 bits como endereço do escravo, ou seja, passa a ser possível definir valores acima de 255 e abaixo de 65535 como endereço do escravo. Neste caso, o endereço do escravo passa a ser definido no protocolo por três bytes. Além disto, a opção Default Slave Address passa a não funcionar mais.</p>

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

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

Data Address Model Offset

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

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

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

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

padrão utilizado. Em caso de dúvida, consulte o suporte do fabricante.

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

Outras Opções

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

Outras opções disponíveis na aba Modbus

OPÇÃO	DESCRIÇÃO
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 <i>Slave Id</i> (parâmetro <i>N1/B1</i> na configuração numérica ou o campo Dispositivo na configuração por Strings) em 1000, ou seja, todos os Tags com <i>Slave Id</i> igual a 1000 têm este valor substituído pelo valor configurado na caixa de edição Default Slave Address . Também é possível forçar o uso do endereço padrão em todos os Tags, independente do valor de <i>Slave Id</i> configurado, selecionando-se a opção Use Default Address .
Wait Silence on error	Se esta opção estiver habilitada, após cada erro de comunicação o Driver permanece em <i>loop</i> , recebendo dados até que ocorra um <i>time-out</i> . Isto limpa o canal de recepção e impede que ocorram problemas em futuras comunicações devido à recepção de bytes atrasados que ainda estejam trafegando no momento do erro, e que possam ser confundidos com uma resposta a um novo comando.

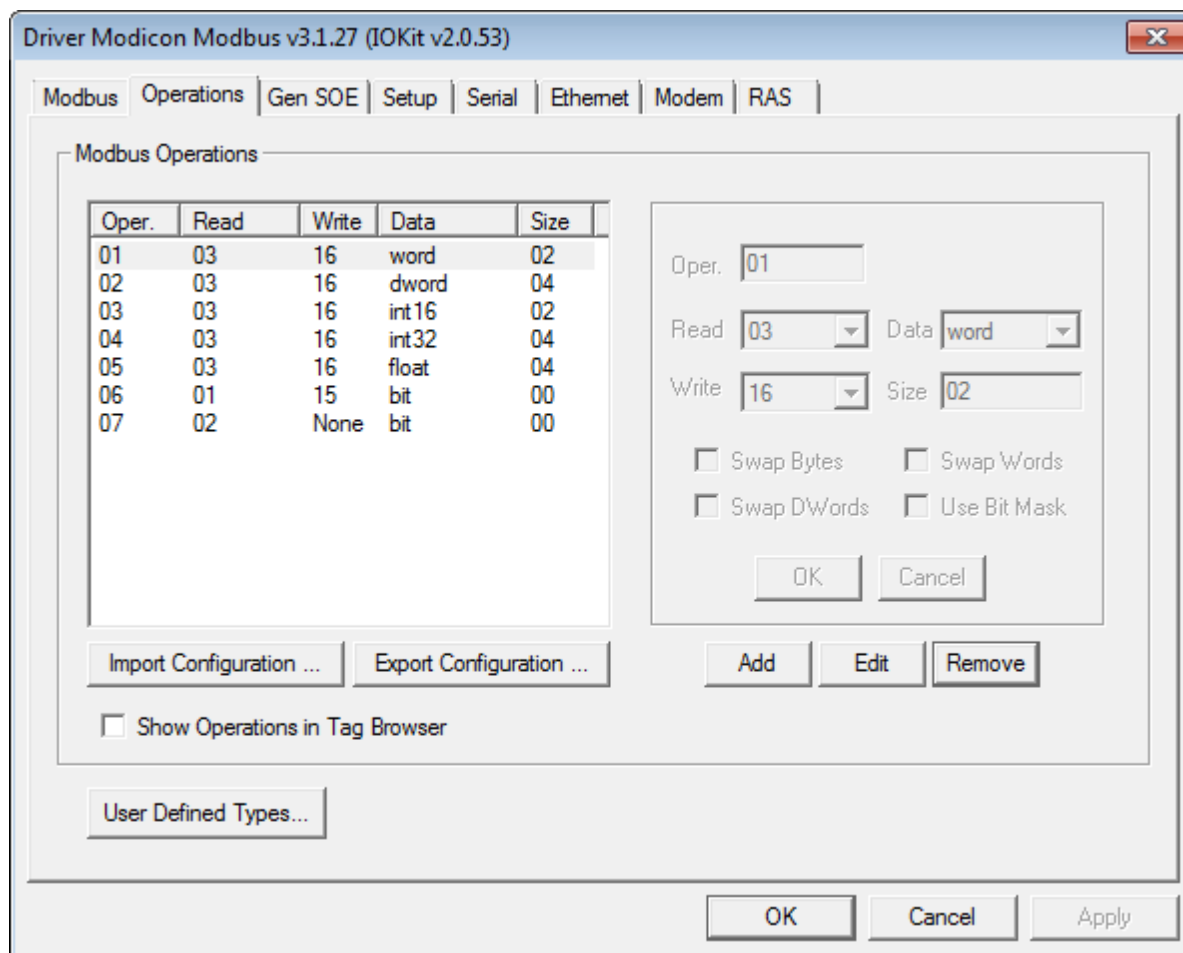
OPÇÃO	DESCRIÇÃO
Reconnect after Timeout (Ethernet only)	Com esta opção habilitada, após qualquer erro de <i>time-out</i> na recepção de <i>frames</i> do equipamento, o Driver promove a desconexão e a reconexão da camada física, limpando a conexão de possíveis <i>frames</i> atrasados que estejam em tráfego e que possam afetar futuras requisições. Esta opção deve sempre ser habilitada caso seja inevitável o uso de 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 <i>frame</i> de resposta correto de um outro <i>frame</i> atrasado resultante de leitura anterior, possivelmente de outro endereço, e que tenha falhado por <i>time-out</i> . Para novos projetos, recomenda-se fortemente que 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 do IOKit para que o Driver se reconecte após o <i>time-out</i> . Caso contrário, o <i>time-out</i> apenas gera uma desconexão e cabe à aplicação o gerenciamento desta nova conexão.

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

Aba Operations

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

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



Aba Operations na janela de configurações do Driver

Operações

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

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

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

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

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

Funções

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

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

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

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

Formatação de Dados

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

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

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

Tipos de Dados Definidos pelo Usuário

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

Byte Order

Além das funções de leitura e escrita do protocolo e do tipo de dado utilizado, cada operação permite também atribuir manipulações adicionais aos bytes, relacionadas ao chamado *byte order*, ou seja, a ordem dos bytes dentro de cada valor. São as chamadas opções de *swap* (*Swap Bytes*, *Swap Words* e *Swap DWords*).

Tais opções somente necessitam ser habilitadas no caso de equipamentos que não respeitem a ordem de bytes padrão do protocolo.

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

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

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

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

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

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

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

Máscara de Bits

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

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

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

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

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

Operações Padrão do Driver

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

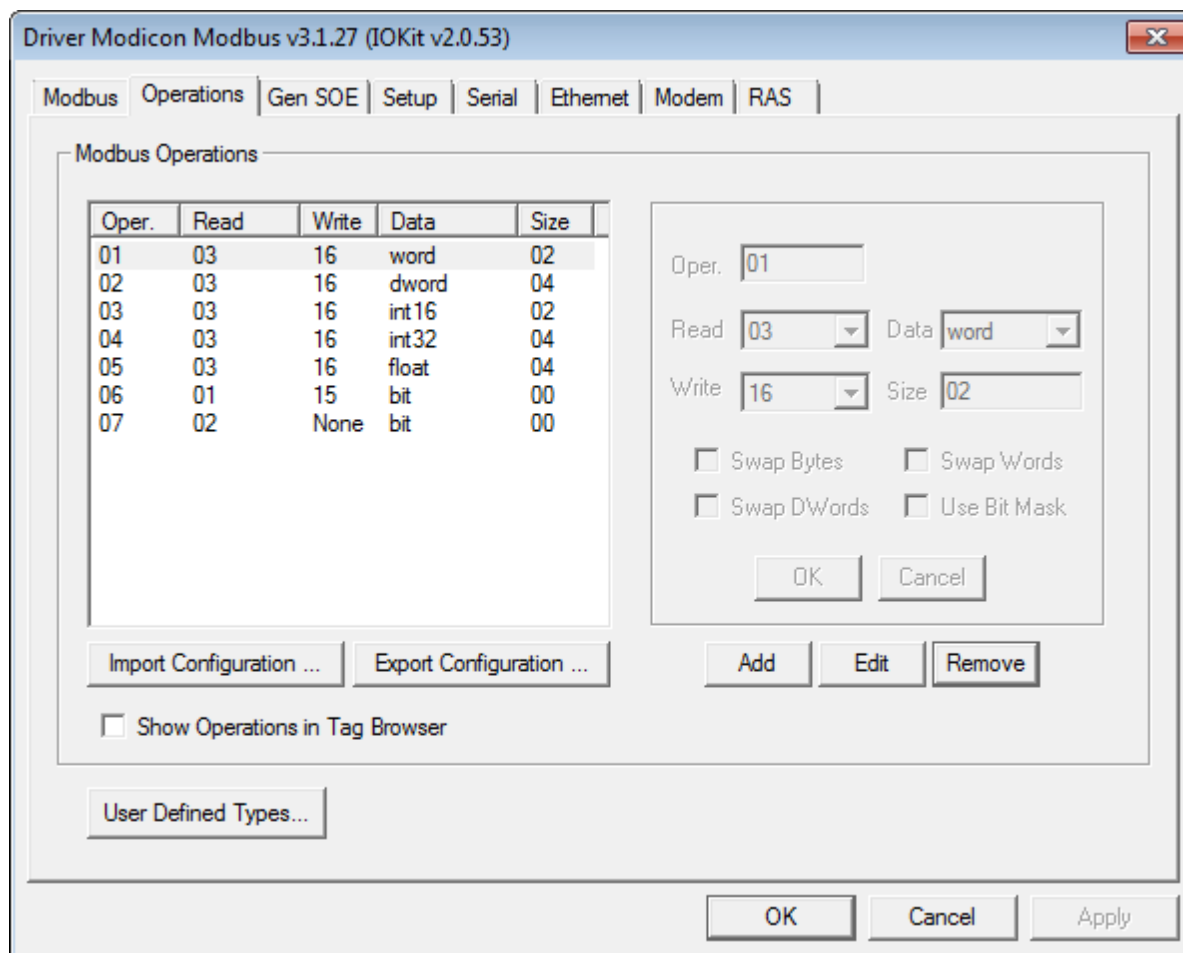
Operações padrão

OPERAÇÃO	FUNÇÃO DE LEITURA	FUNÇÃO DE ESCRITA	TIPO DE DADOS	FINALIDADE
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 e escrita de valores de ponto flutuante de 32 bits
6	3 - Read Holding Registers	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 (<i>Discrete Inputs</i>)

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

Definição de Novas Operações

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



Adicionando uma nova operação

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

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

Opções de campos para operações

OPÇÃO	DESCRIÇÃO
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 <i>frame</i> de uma determinada função.

OPÇÃO	DESCRIÇÃO
Swap Bytes	Indica que o Driver deve inverter a ordem dos bytes, um a um, para obter o valor.
Swap Words	Indica que o Driver deve inverter a ordem dos bytes, dois a dois (em Words), para obter o valor.
Swap DWords	Indica que o Driver deve inverter a ordem dos bytes, quatro a quatro (em DWords), para obter o valor.
Use Bit Mask	Habilita o mascaramento de bits de registradores, através do parâmetro <i>N3/B3</i> . Esta opção afeta apenas a leitura e pode ser usada apenas com tipos de dados inteiros, com ou sem sinal, com pelo menos 16 bits de tamanho (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 da aplicação, deixando esta opção desmarcada (veja 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 2.0 do Driver Modbus Master/Slave, que armazenavam estas configurações em um arquivo modbus.ini. Este Driver não utiliza mais arquivos INI para armazenar tais configurações, que agora são armazenadas no próprio arquivo da aplicação. Para maiores detalhes, consulte o tópico Importação e Exportação de Operações .
Export Configuration	Esta opção executa a operação inversa da anterior, gerando um arquivo INI contendo as configurações de operações, no formato atual ou no mesmo formato das versões anteriores deste Driver. Desta forma, é possível guardar em um arquivo as configurações de operações de um determinado equipamento, que podem ser utilizadas em outras aplicações. Para maiores detalhes, consulte o tópico Importação e Exportação de Operações .

OPÇÃO	DESCRIÇÃO
Show Operations in Tag Browser	Se esta opção não estiver selecionada (padrão), são mostrados modelos de Tags configurados por Strings (campos Dispositivo e Item) no Tag Browser. Se estiver selecionada , modelos de Tags configurados numericamente (parâmetros <i>N/B</i>), representando as operações configuradas, são mostrados no Tag Browser. Quando novas instâncias do Driver são criadas, esta opção vem desmarcada por padrão. Em aplicações legadas, quando a versão do Driver é atualizada a partir de uma versão anterior à 3.1, a opção já vem selecionada, mantendo o comportamento das versões anteriores.
Add	Adiciona uma nova operação à lista.
Edit	Atualiza a operação selecionada na lista (equivale a clicar duas vezes no 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 do Driver corresponde ao padrão do protocolo, sendo esta a opção recomendada para a maioria dos equipamentos.**

Tipos de Dados Suportados

A tabela a seguir relaciona os tipos de dados nativos do 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 do Driver são convertidos para **Word** no nível do protocolo, para a leitura ou escrita no equipamento ou dispositivo escravo.

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

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

Opções disponíveis para tipos de dados

TIPO	FAIXA	DESCRIÇÃO
Char	-128 a 127	Palavra de oito bits, caractere. A escrita deve ocorrer sempre em blocos de tamanho par (Words)
Byte	0 a 255	Palavra de oito bits sem sinal. A escrita deve ocorrer sempre em blocos de tamanho par (Words)

TIPO	FAIXA	DESCRIÇÃO
Int8	-128 a 127	Palavra de 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 UInt)	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)
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 oito bits $2^{[-128 \dots +127]}$ e 24 bits de mantissa $[-2^{23} \dots + (2^{23} - 1)]$. (quatro bytes: EXP F2 F1 F0). Para mais informações, consulte a documentação do equipamento
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 oito bits (Chars)

TIPO	FAIXA	DESCRIÇÃO
BCD	Ver descrição	Valor numérico BCD (<i>Decimal Codificado em Binário</i>). Ao utilizar este tipo de dados, a aplicação deve fornecer um valor decimal positivo e inteiro, a ser enviado no formato 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 o valor. Uma vez que na codificação BCD cada algarismo é convertido em um <i>nibble</i> , tem-se que os valores permitidos devem possuir um número máximo de algarismos igual ao dobro do valor especificado no campo Size , ou seja, se for selecionado o valor dois para o campo Size , o máximo valor que pode ser enviado é 9999. Já se Size é igual a quatro, o valor máximo é 99999999. Os valores permitidos para o campo Size no caso de tipos de dados BCD são dois (Word) e quatro (Double Word). Para maiores detalhes sobre a codificação BCD, consulte o tópico Codificação BCD
GE_events	Ver descrição	Tipo de dados utilizado na leitura do <i>buffer</i> de eventos (SOE) de CLP GE PAC RX7. Sua definição é permitida apenas em operações que utilizem a função especial de leitura GE SOE . Estes eventos são retornados como blocos de dois Elementos, com <i>timestamps</i> definidos pelo controlador. Para mais informações, veja 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 um bit

TIPO	FAIXA	DESCRIÇÃO
SP_events	Ver descrição	Tipo de dados utilizado na leitura de eventos (SOE) de relés Schneider Electric das séries SEPAM 20, 40 e 80. Sua definição só é permitida quando a operação utilizar como função de leitura a função especial SP SOE . Estes eventos são retornados como um Bloco de três Elementos, com <i>timestamp</i> fornecido pelo equipamento. Para mais informações, veja 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 (ver nota a seguir)	Tipo de dados de data e hora composto por uma estrutura de oito bytes, criado originalmente para ser utilizado na leitura de eventos que usam o 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 internamente como uma estrutura de Words , a única função de <i>swap</i> válida para este tipo de dados é Swap Bytes . A representação deste tipo de dados na memória do CLP 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 (ver nota a seguir)	Tipo de dados de data e hora composto por uma estrutura de oito bytes, utilizado por relés Schneider Electric das séries SEPAM 20, 40 e 80, geralmente para representar um <i>timestamp</i> . Para mais informações, consulte a documentação do equipamento
UTC64d	1/1/1970 00:00 a 31/12/2035 23:59:59.999 (ver nota a seguir)	Tipo de dados de data e hora representado em formato Double (IEEE 754 64 bits) , com os segundos desde 1/1/1970 00:00

TIPO	FAIXA	DESCRIÇÃO
UTC32	1/1/1970 00:00 a 31/12/2035 23:59:59.999 (ver nota a seguir)	Tipo de dados de data e hora representado em formato inteiro sem sinal de 32 bits (DWord ou UInt), com os segundos desde 1/1/1970 00:00. Neste formato não são representados os milissegundos, sendo considerados sempre 0 (zero)

NOTA: Embora a representação em si dos tipos de dados de data e hora da tabela anterior possa representar datas superiores a 31/12/2035, este limite é mostrado na tabela pelo fato de os aplicativos da Elipse Software não possuírem suporte, atualmente, para faixas de valores superiores a este limite em estampas de tempo (*timestamps*).

Tipo de Dados GenTime

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

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

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

Estrutura dos registradores

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

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

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

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

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

Tipos de Dados Definidos pelo Usuário

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

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

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

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

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

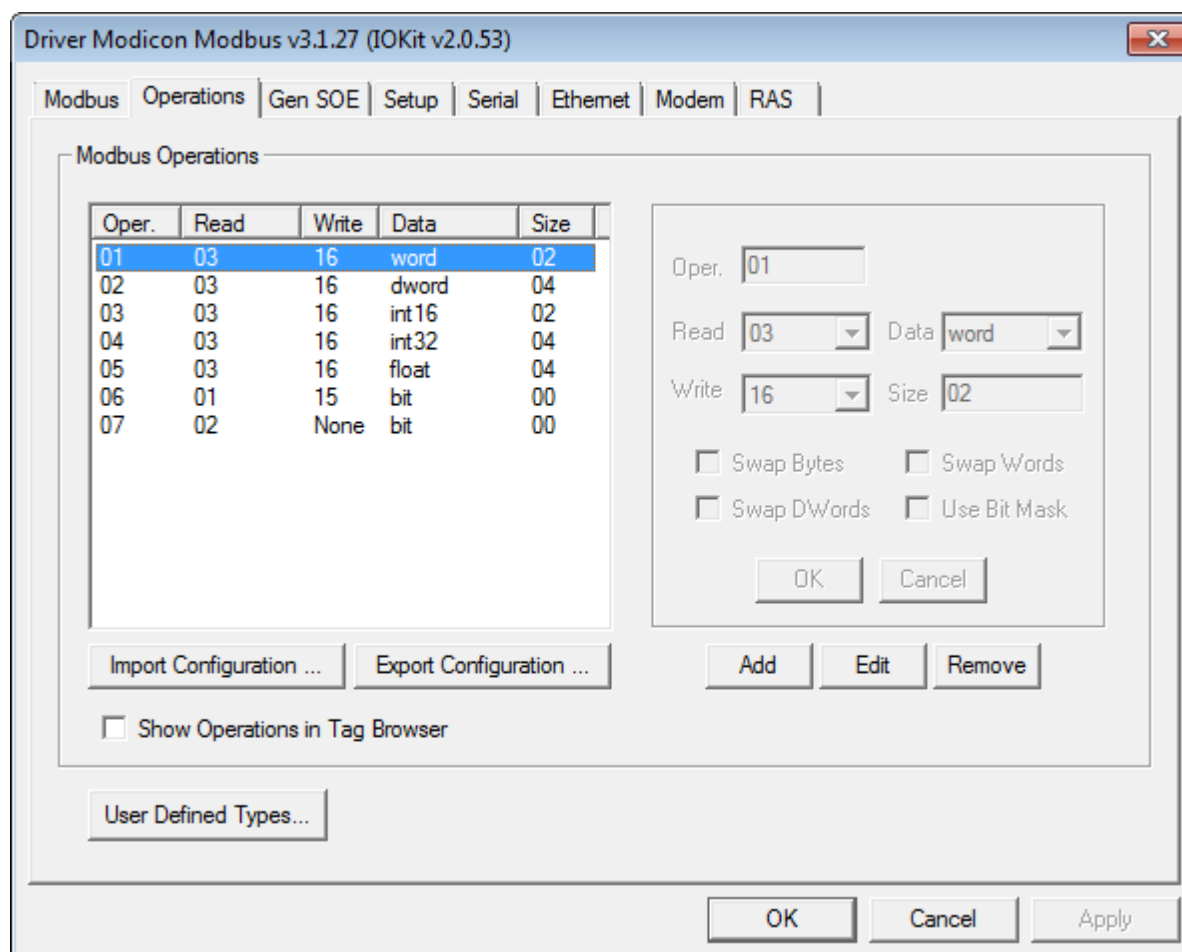
Aplicações

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

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

Configuração de Tipos de Dados Definidos pelo Usuário

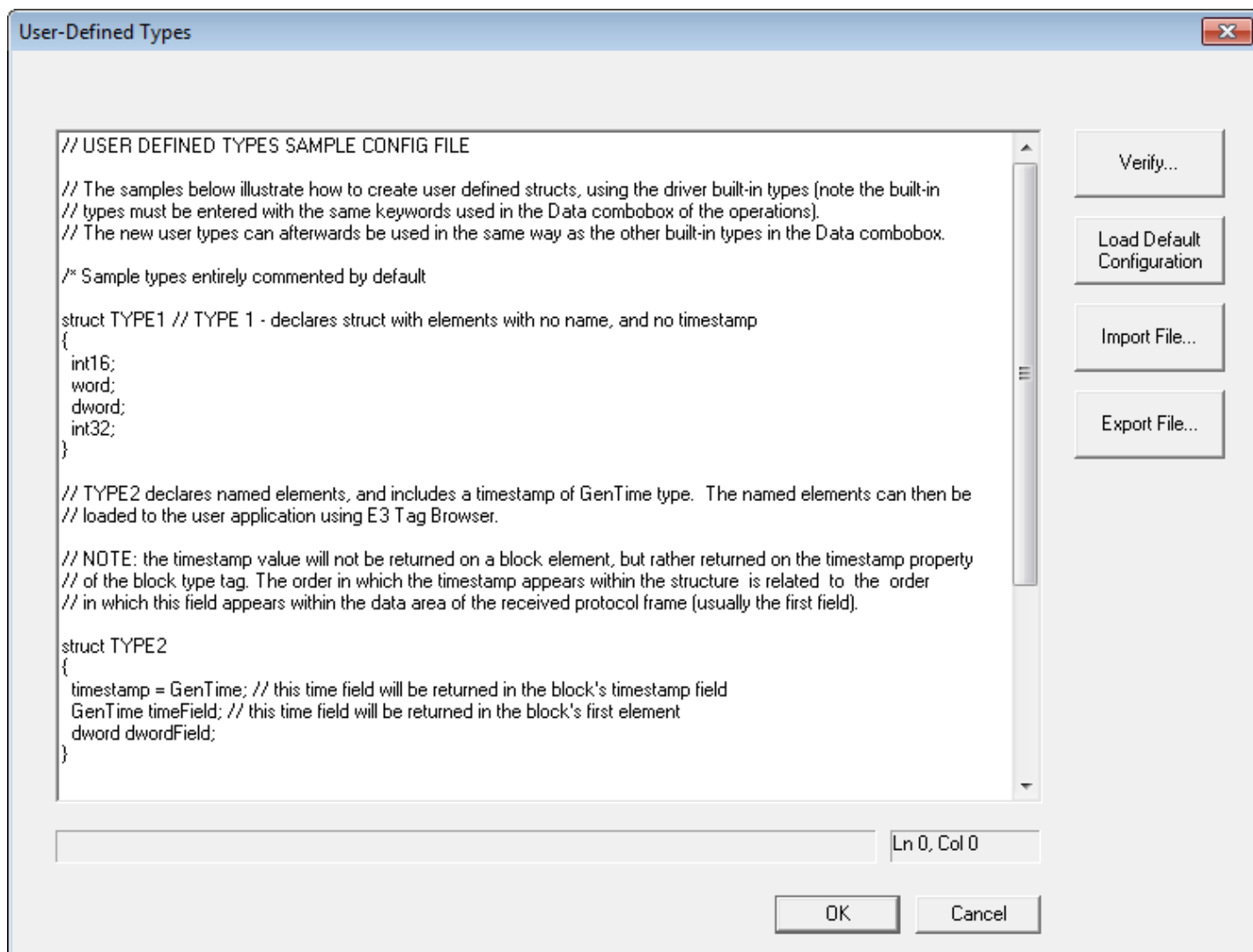
A configuração de tipos de dados definidos pelo usuário é realizada em janela específica, clicando em **User Defined Types** na **Aba Operations** da janela de configurações do Driver, conforme a figura a seguir.



Aba Operations da janela de configurações do Driver

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

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



Configuração de tipos de dados definidos pelo usuário

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

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

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

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

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

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

```

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

```

Onde:

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

Importação e Exportação

As opções **Import File** e **Export File** permitem importar e exportar o arquivo de configuração de tipos de dados definidos pelo usuário para arquivos texto em disco. Podem ser utilizados para a realização de cópias de segurança do arquivo, ou para compartilhá-lo entre vários Drivers. O arquivo é sempre gravado e lido no

formato ANSI padrão do Windows (Charset **Windows-1252**). Futuras versões do Driver podem suportar outros formatos.

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

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

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

Utilizando Tipos de Dados Definidos pelo Usuário na Configuração por Strings

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

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

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

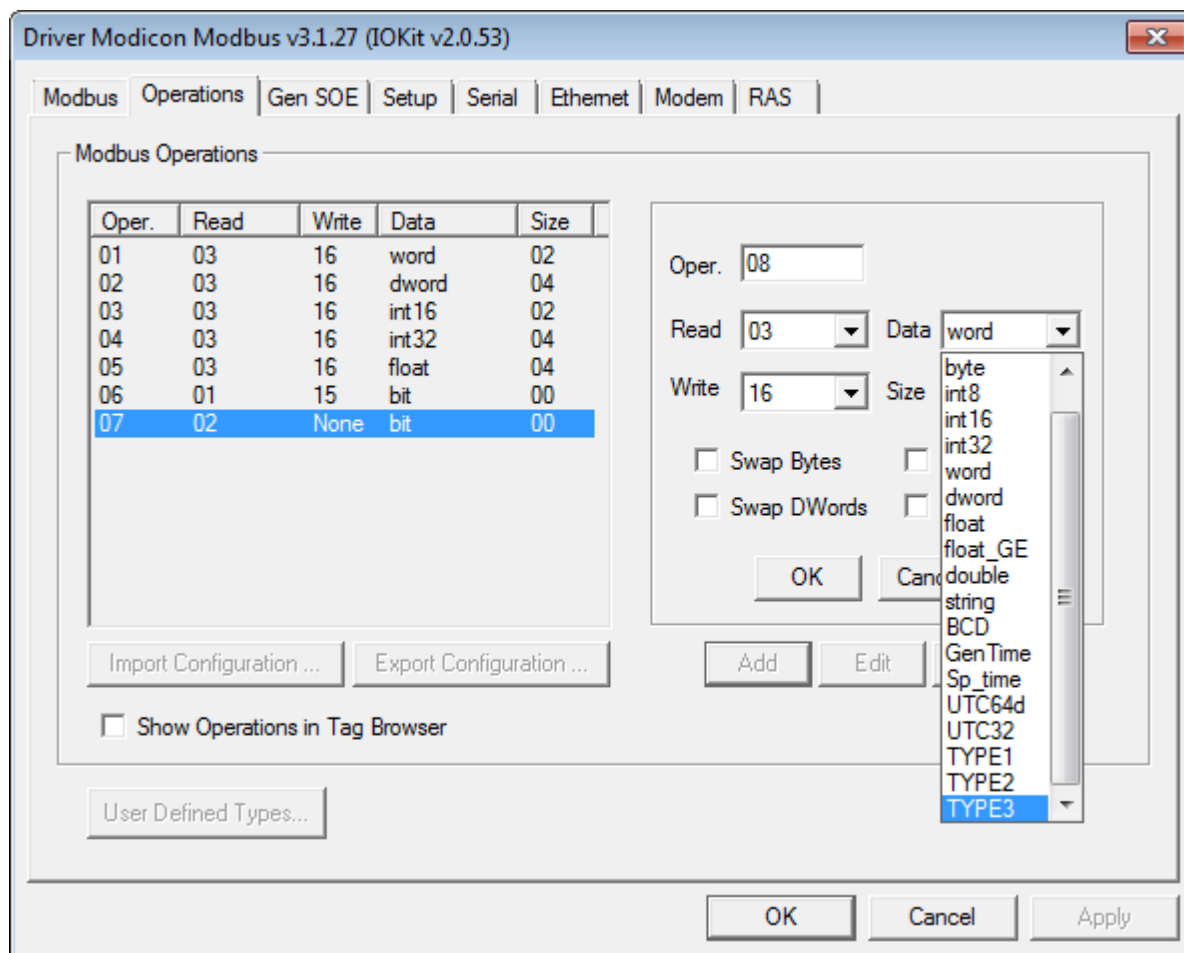
- Leitura ou escrita de *Holding Registers* (funções **03** e **06**) de endereço 100 do equipamento com *Id* 5, interpretado como um tipo de dados definido pelo usuário chamado "mytype", com *Slave Id* no campo **Item**:
 - **Dispositivo:** "" (empty String)
 - **Item:** "5:shr100.mytype"

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

Utilizando Tipos Definidos pelo Usuário na Configuração Numérica

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

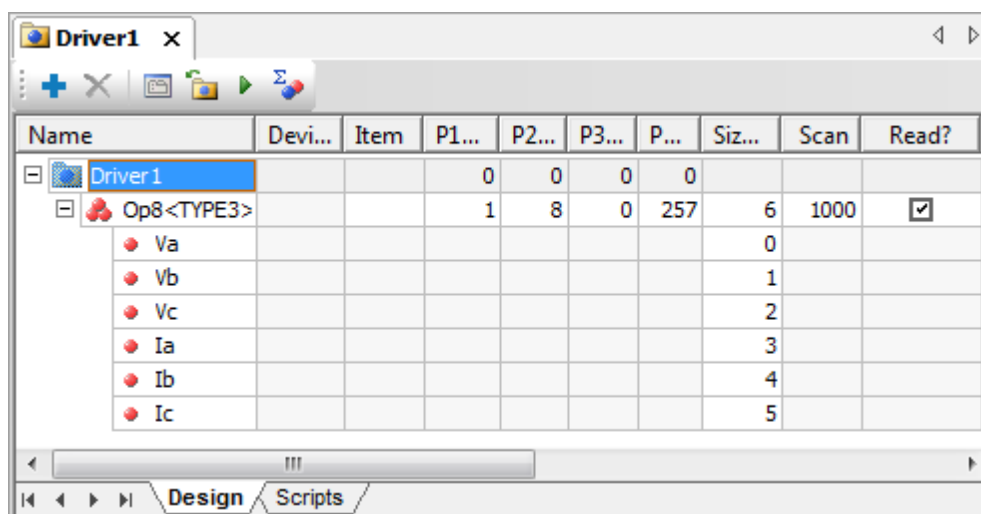
A figura a seguir mostra a configuração de uma nova operação que utiliza o tipo de dados definido pelo usuário (estrutura) de nome **TYPE3**, mostrado no exemplo acima, após o usuário clicar em **Add**.



Adicionar tipo de dados definido pelo usuário

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

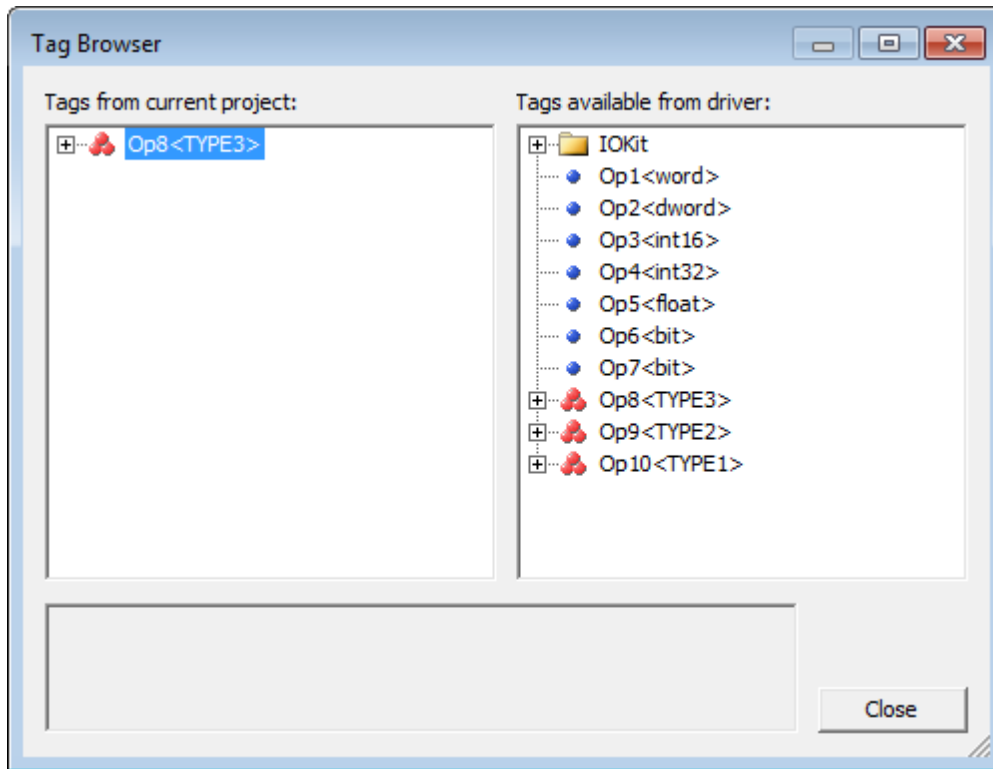
Após a definição da nova operação, usando o novo tipo **TYPE3**, defina um Tag Bloco com o mesmo tipo de dados e com tamanho igual ao número de elementos da estrutura, conforme mostrado na figura a seguir.




Declaração de Tags usando estruturas no E3 ou Elipse Power

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

a seguir ilustra o procedimento.



Uso do Tag Browser para definir Tags usando estruturas

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

Leitura Reportada a Eventos

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

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

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

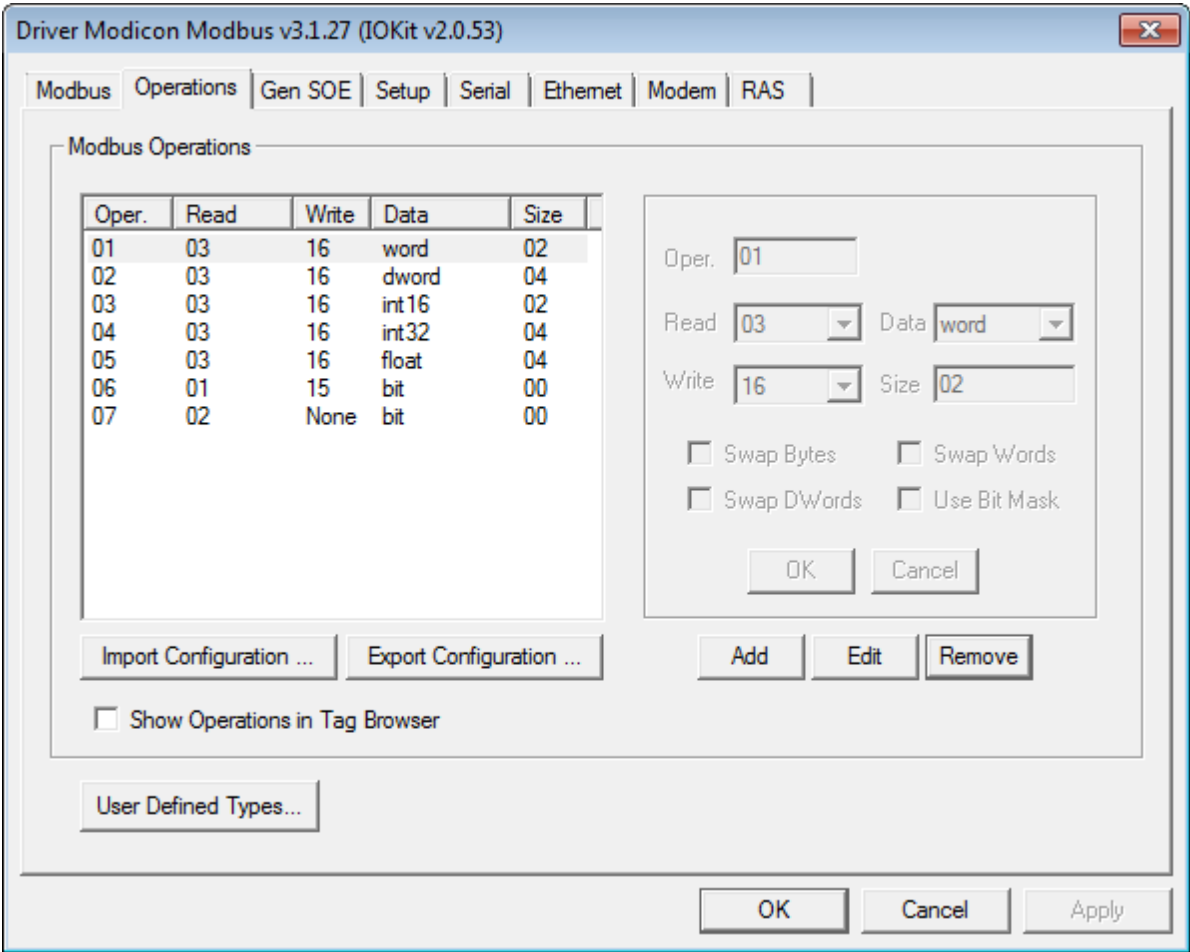
No caso de funções especiais de SOE, como a **função Gen SOE**, o retorno reportado a eventos é definido pelo próprio algoritmo proprietário da função.

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

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

Importação e Exportação de Operações

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



Opções de importação e exportação de operações

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

Nas versões anteriores à 2.00 deste Driver, a configuração das operações era executada no arquivo

modbus.ini, que era carregado no momento da inicialização do objeto Driver. Os arquivos modbus.ini destas versões antigas ainda podem ser carregados na versão atual do Driver, utilizando a opção de importação.

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

Importação

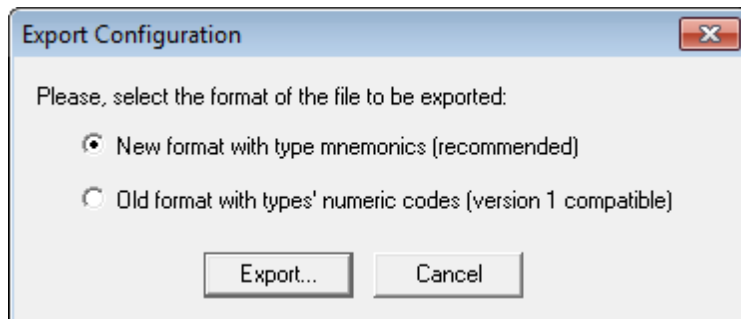
A importação de arquivos de configuração é bastante simples. Clique em **Import Configuration** e selecione o arquivo INI. O Driver deve carregar as configurações de operações, que aparecem imediatamente no quadro **Modbus Operations**. Este Driver permite a importação de arquivos gerados pelas suas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

Exportação

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

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

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



Opções de exportação

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

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

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

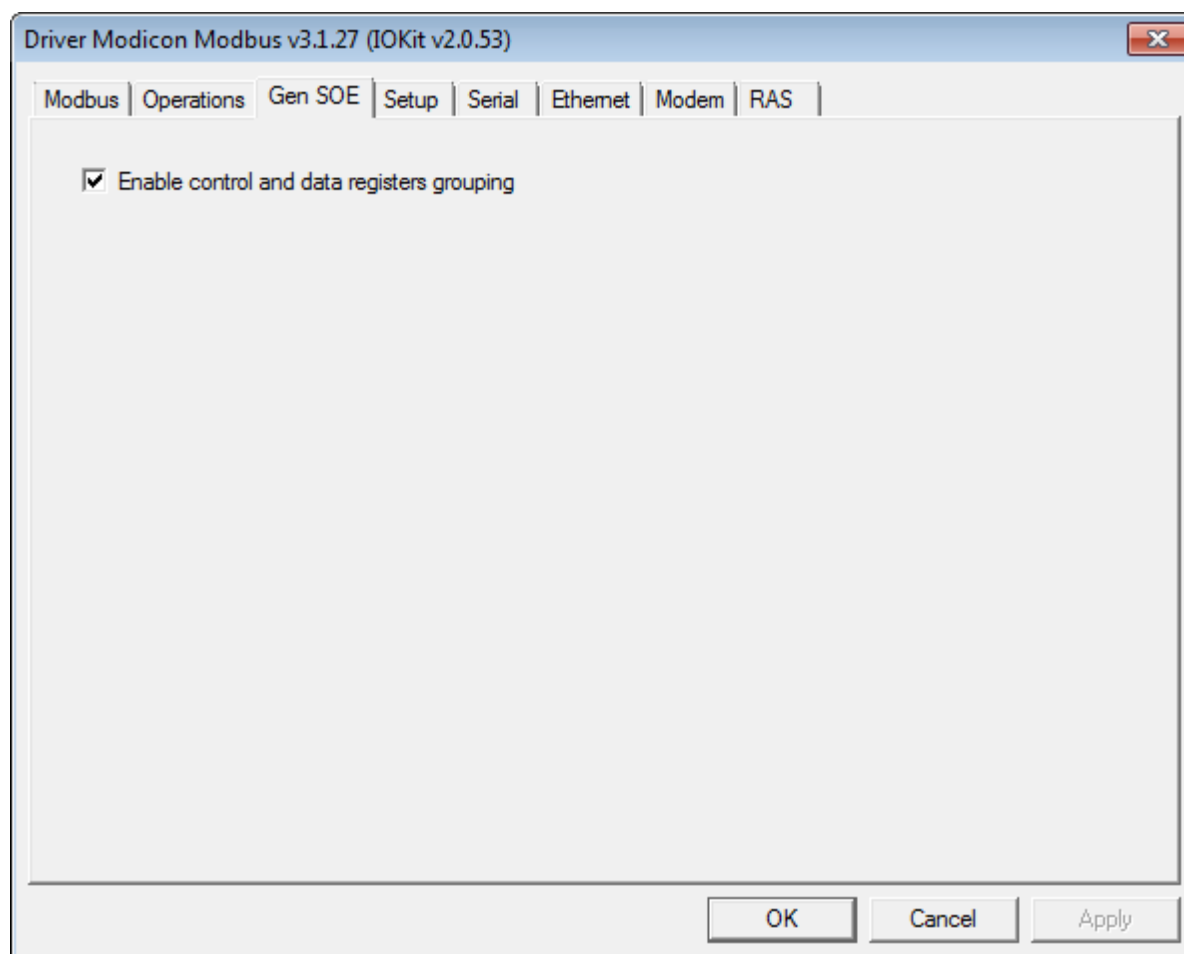
Note que, para exportar com sucesso arquivos modbus.ini a serem carregados em versões anteriores à versão 2.00, as operações não podem definir nenhum tipo de dados novo que não tenha sido implementado pela versão de destino, e nem tampouco ter operações que utilizem o parâmetro **Use bit mask**, caso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

De maneira geral, recomenda-se evitar a exportação de configurações para versões anteriores, priorizando

sempre a atualização do Driver.

Aba Gen SOE

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



Aba Gen SOE

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

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

Configuração em Modo Offline

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

Parâmetros disponíveis

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

Para mais informações sobre a configuração **Offline** em tempo de execução, consulte o **Manual do Usuário do IOKit**.

Configurando Tags

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

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

Configurando um Tag de Comunicação

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

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

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

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

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

Configuração por Strings

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

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

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

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

Leitura em Bloco

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

Se o equipamento suporta **superblocos**, caso o supervisor não seja o Elipse SCADA, que não suporta este recurso, sugere-se criar todos os Tags como Tags simples, mantendo a opção **EnableReadGrouping** do Driver em Verdadeiro, deixando o agrupamento a cargo do algoritmo de superblocos.

Campo Dispositivo

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

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

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

Campo Item

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

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

Onde os parâmetros entre colchetes são opcionais.

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

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

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

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

1. Leitura ou escrita de *Holding Registers* (funções **03** e **16**) de endereço 100 do equipamento com *Id* 1 e com *Slave Id* no campo **Dispositivo**:
 - a. **Dispositivo**: "1:"
 - b. **Item**: "hr100"
2. Leitura ou escrita de *Holding Register* (funções **03** e **16**) de endereço 120 do equipamento com *Id* 3 e com *Slave Id* no campo **Item**:
 - a. **Dispositivo**: "" (**String** vazia)
 - b. **Item**: "3:hr120"
3. Leitura ou escrita de *Coil* (funções **01** e **15**) de endereço 65535 (FFFFh) com *Slave Id* 4, aqui especificado no campo **Item**:
 - a. **Dispositivo**: "" (**String** vazia)
 - b. **Item**: "4:cl&hFFFF" (ou opcionalmente "4:cl65535")

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

Name	Device	Item	P1/N1...	P2/N2...	P3/N3...	P4/N4...
Driver1			0	0	0	0
Analog						
Temperature_1	1:	hr1	0	0	0	0
Temperature_2	1:	hr2	0	0	0	0
Level_T1	1:	hr3	0	0	0	0
Level_T2	1:	hr4	0	0	0	0
Digital						
Status_B1	1:	sd1	0	0	0	0
Status_B2	1:	sd2	0	0	0	0
Status_B3	1:	sd3	0	0	0	0
Status_B4	1:	sd4	0	0	0	0
Status_B5	1:	sd5	0	0	0	0
Status_B6	1:	sd6	0	0	0	0
AutoMan_B1	1:	sd7	0	0	0	0
AutoMan_B2	1:	sd8	0	0	0	0
AutoMan_B3	1:	sd9	0	0	0	0
AutoMan_B4	1:	sd10	0	0	0	0
AutoMan_B5	1:	sd11	0	0	0	0
AutoMan_B6	1:	sd12	0	0	0	0
Failure_B1	1:	sd13	0	0	0	0
Failure_B2	1:	sd14	0	0	0	0
Failure_B3	1:	sd15	0	0	0	0
Failure_B4	1:	sd16	0	0	0	0
Failure_B5	1:	sd17	0	0	0	0
Failure_B6	1:	sd18	0	0	0	0

Tags do Tutorial do E3 configurados por Strings

Campos Obrigatórios e Opcionais

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

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

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

1. **Tipo:** Define como os bytes da área de dados do *frame* de comunicação devem ser interpretados. Se for omitido, são usados os tipos padrão definidos pelo protocolo para as funções em uso, ou seja, **Word** para

funções de acesso a registradores e **Bit** para funções de acesso a dados digitais (*Coils* e *Discrete Inputs*). Consulte a **tabela de mnemônicos** dos tipos suportados, mais adiante neste tópico.

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

Exceções

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

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

Exemplo:

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

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

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

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

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

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

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

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

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

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

Espaço de Endereçamento

Ao invés de definir explicitamente as funções Modbus ou especiais de leitura e escrita a serem utilizadas, como ocorre na antiga **configuração numérica** e seu conceito de **operações**, na configuração por **Strings** o

usuário define um *espaço de endereçamento* através de um dos mnemônicos listados na tabela a seguir, já associado a funções pré-definidas do protocolo e seus respectivos tipos nativos de dados.

Espaços de endereçamento e mnemônicos

ESPAÇO DE ENDEREÇAMENTO	MNEMÔNICO	TIPO NATIVO	FUNÇÕES DE LEITURA OU ESCRITA	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 <i>overhead</i> .
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 <i>overhead</i>
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 de Massa	abbmge	Word (16 bits)	Funções 65 03 e None (somente leitura)	
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.rstmxm	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	

ESPAÇO DE ENDEREÇAMENTO	MNEMÔNICO	TIPO NATIVO	FUNÇÕES DE LEITURA OU ESCRITA	COMENTÁRIOS
SEPAM SOE Events	spsoe	SP_events	Função de leitura SP SOE e de escrita None	Coleta do medidor e retorna ao Tag uma estrutura (tipo SP_events) com eventos de quaisquer pontos (veja o tópico SEPAM SOE)
SEPAM SOE Single Point Events	ptspsoe	Int32	Função de leitura SP SOE e de escrita None	Coleta do medidor e retorna ao Tag um valor inteiro do campo Edge , relativo a eventos de um ponto específico (veja 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 os equipamentos que usam Modbus, deve-se especificar o tipo de dados a ser usado, conforme explicado nesta seção.

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

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

Tipos de dados suportados

TIPO	MNEMÔNICO	ALIAS
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

TIPO	MNEMÔNICO	ALIAS
Double ou Real	double	d
String	string	s
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 (veja o tópico **Tipos de Dados Definidos pelo Usuário**).

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

Exemplos

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

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 **Double** exige a leitura de quatro endereços de *Holding Registers*. Pelos mesmos motivos, a leitura e escrita de Tags de "hr" de tipo **Byte** só pode ser realizada aos pares. No equipamento, cada endereço de *Holding Register* contém sempre dois bytes.

Tamanho do Tipo de Dados

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

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

1. Leitura ou escrita de *Holding Registers* (funções **03** e **16**) de endereço 100 do equipamento com *Id* 1, interpretado como uma **String** de 10 bytes (cinco registros "hr"), com *Slave Id* no campo **Dispositivo**:

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

b. **Item:** "hr100.s10"

2. Leitura ou escrita de *Holding Registers* (funções **03** e **16**) de endereço 100 do equipamento com *Id* 1, interpretado como **BCD** de oito dígitos (quatro bytes ou dois registros "hr"), com *Slave Id* no campo **Item**:

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

b. **Item:** "1:hr100.bcd4"

Ordenamento de Bytes (Byte Order)

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

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

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

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

interpretação.

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

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

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

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

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

Operações de permuta ou swap

	SWAP BYTES	SWAP WORDS	SWAP DWORDS	ALIAS	ALIAS 2 (SWAPS)	ORDENAMENTO *
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 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

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

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

Já "b1" realiza a permuta dos bytes, dois a dois, ou seja, ao receber um **Word** (inteiro sem sinal de 16 bits)

com o valor em hexadecimal 0102h, o valor que é retornado ao Tag é 0201h, com os bytes trocados. Equivale à antiga opção **Swap Bytes**.

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

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

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

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

Como Selecionar a Opção de Ordenamento Correta?

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

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

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

Podem ocorrer basicamente três situações:

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

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

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

1. Leitura ou escrita de *Holding Registers* (funções **03** e **16**) de endereço 1500 do equipamento com *Id* 5, interpretado como um **Double** sem inversão de bytes, com *Slave Id* no campo **Item**:
 - a. **Dispositivo**: "" (**String** vazia)
 - b. **Item**: "5:hr1500.d", ou "5:hr1500.double" ou ainda "5:hr1500.d.b0"
2. Leitura ou escrita de *Holding Registers* (funções **03** e **16**) de endereço 1500 do equipamento com *Id* 5, interpretado como um **Double** com o byte menos significativo de cada **Word** vindo antes e com *Slave Id* no campo **Item**:
 - a. **Dispositivo**: "" (**String** vazia)
 - b. **Item**: "5:hr1500.d.b1" ou "5:hr1500.double.b1", ou ainda "5:hr1500.double.sb"
3. Leitura ou escrita de *Holding Registers* (funções **03** e **16**) de endereço 1500 do equipamento com *Id* 5, interpretado como um **Double** com os bytes menos significativos vindo antes (*little endian*) e com *Slave Id* no campo **Item**:
 - a. **Dispositivo**: "" (**String** vazia)
 - b. **Item**: "5:hr1500.d.b7" ou outras variações, como "5:hr1500.d.lsb" e "5:hr1500.d.sb.sw.sdw"

Tags Especiais do Driver

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

Tags especiais

DEVICE	ITEM	OPERAÇÃO
	ForceWaitSilence	Escrita
<slave id>:	LastExceptionCode	Leitura ou escrita

Configuração Numérica

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

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

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

Operações

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

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

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

Parâmetros de Configuração dos Tags de Comunicação

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

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

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

- **N2/B2:** Código da operação. Referencia uma operação adicionada na janela de configurações do Driver (veja o tópico **Aba Operations**).
- **N3/B3:** Parâmetro adicional. Este parâmetro em geral não é usado e pode ser deixado em 0 (zero). Só é usado em quatro situações:
 - **Funções Modbus 20 e 21:** No caso de operações que utilizem estas funções de acesso a arquivo (funções **20 e 21**), o parâmetro *N3/B3* especifica o arquivo a ser acessado.
 - **Use Bit Mask:** Para Tags referenciando operações com a opção **Use Bit Mask** habilitada, o parâmetro *N3/B3* especifica o número do bit a ser acessado (veja o tópico **Aba Operations**).
 - **Tipos de Dados Definidos pelo Usuário:** Para operações que usem estruturas, se o parâmetro *B3* for maior que 0 (zero), define o retorno de um *array* de blocos reportado a eventos, por meio de uma sequência de eventos **OnRead** do Tag (veja o tópico **Tipos de Dados Definidos pelo Usuário**).
 - **Função Especial Gen SOE:** Em operações que utilizem a função especial de leitura **Gen SOE**, o parâmetro *N3/B3* indica o tamanho da tabela associada na memória do CLP ou dispositivo escravo, em número máximo de eventos comportados (veja o tópico **Algoritmo de Leitura de SOE Genérico da Elipse Software**).
- **N4/B4:** Endereço do registrador, variável ou bit no equipamento ou dispositivo escravo (CLP) em que se deseja ler ou escrever, conforme o mapa de registradores do equipamento (verifique o manual do equipamento). É importante ajustar corretamente a opção **Data Address Model Offset** (veja o tópico **Aba Modbus**) e verificar se a documentação do fabricante não usa *offsets* utilizados por equipamentos Modbus antigos, da chamada **Modbus Convention**.
- **Tamanho/Índice (somente Tags Bloco):** Cada Elemento de Bloco representa o valor de um dado de tipo definido na operação utilizada (parâmetro *N2/B2*). Note que o protocolo só suporta tipos de dados

Bit ou **Word**. Desta forma, se a operação selecionar o tipo de dados **DWord** (32 bits) para cada Elemento de Bloco, o Driver necessita ler dois registros consecutivos do equipamento.

Tags Especiais

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

Dicas de Endereçamento (Modbus Convention)

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

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

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

Offsets do padrão Modbus Convention

TIPO DE DADOS (STANDARD DATA MODEL)	FUNÇÃO MODBUS	OFFSET
Coils	01: Read Coils (0x) 05: Write Single Coil (0x) 15: Write Multiple Coils (0x)	00000
Discrete Inputs	02: Read Discrete Inputs (1x)	10000
Input Registers	04: Read Input Registers (3x)	30000
Holding Registers	03: Read Holding Registers (4x) 06: Write Single Register (4x) 16: Write Multiple Registers (4x)	40000
File Register (antiga Extended Memory file)	20: Read General Reference (6x) 21: Write General Reference (6x)	60000

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

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

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

Exemplos

ENDEREÇO COM OFFSET (EQUIPAMENTO)	ENDEREÇO NO TAG DE COMUNICAÇÃO (ITEM OU N4/B4)	FUNÇÃO MODBUS
01234	1234	01: Read Coils 05: Write Single Coil 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 16: Write Multiple Registers
45789	5789	03: Read Holding Registers 06: Write Single Register 16: Write Multiple Registers
65789	5789	20: Read General Reference 21: Write General Reference

Partição Automática de Blocos

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

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

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

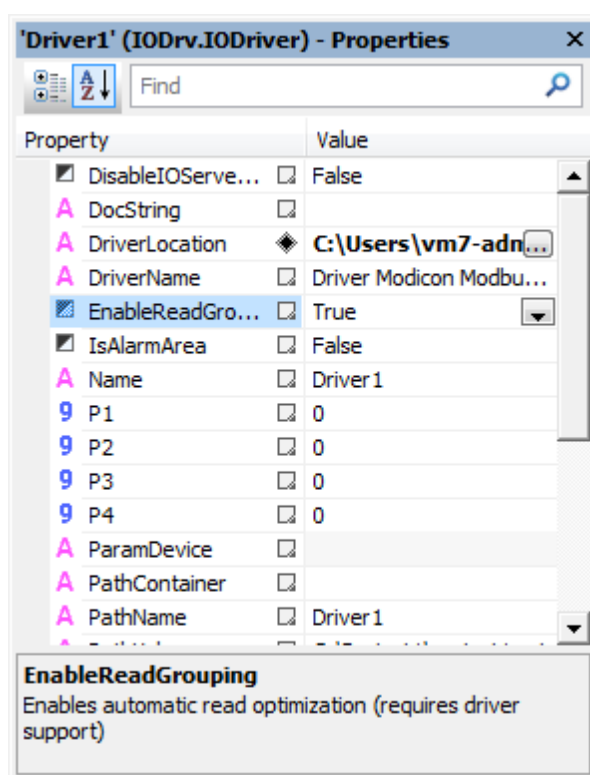
IMPORTANTE: Tanto o agrupamento dos Superblocos no E3 e no Elipse Power como a Partição Automática de Blocos do Driver requerem que o equipamento suporte os limites estabelecidos pelo Modbus padrão (veja o tópico **Limite Máximo para o Tamanho dos Blocos Suportado pelo Protocolo**). Há equipamentos,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

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

Leitura por Superblocos (Agrupamento)

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

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



Propriedade EnableReadGrouping

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

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 E3 como a **Partição Automática de Blocos** do Driver requerem que o equipamento suporte os limites estabelecidos pelo Modbus padrão (veja o tópico **Limite Máximo para o Tamanho dos Blocos Suportado pelo Protocolo**). Há equipamentos, entretanto, que suportam limites inferiores. Para que a divisão automática de blocos e o próprio agrupamento dos Superblocos funcione nestes casos, a partir da versão 2.03, o Driver permite customizar o limite máximo suportado para o PDU (*Protocol Data Unit*). Para isto, na janela de configurações do Driver, aba **Modbus**, habilite a opção **Customize Max. PDU Size** e configure o tamanho máximo de bytes suportado para o PDU pelo equipamento. Caso o equipamento suporte limites diferentes para cada tipo de função, é necessário realizar o agrupamento manual (veja mais adiante neste tópico), observando os limites descritos na documentação do fabricante.

Identificação de Equipamentos que não Suportam Agrupamento Automático (Superblocos)

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

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

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

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

Agrupamento Manual

Em geral, quanto maior for o agrupamento das variáveis em blocos, menor é o número de requisições de leitura necessárias para completar o ciclo de varredura (*scan*) dos Tags da aplicação, aumentando assim a velocidade de atualização dos Tags. Por este motivo, se não for possível usar o agrupamento automático (Superblocos), é preferível criar Tags Bloco contendo o maior número possível de variáveis ao invés de criar

Tags simples (Tags PLC no Elipse SCADA).

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

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

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

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

Configurando Tags Especiais

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

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

Forçar Wait Silence

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

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

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

Configuração por Strings

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

Configuração Numérica

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

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

Leitura do Código da Última Exceção

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

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

Códigos de Exceção

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

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

Configuração por Strings

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

Configuração Numérica

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

Valores dos Elementos de Bloco retornados:

- **Elemento 1 (índice 0):** Código da exceção retornada pelo equipamento (veja o tópico **Lista de Exceções Padrão do Protocolo**)

- **Elemento 2 (índice 1):** Parâmetro *N2/B2* do Tag de Comunicação que gerou a exceção
- **Elemento 3 (índice 2):** Parâmetro *N3/B3* do Tag de Comunicação que gerou a exceção
- **Elemento 4 (índice 3):** Parâmetro *N4/B4* do Tag de Comunicação que gerou a exceção
- **Elemento 5 (índice 4):** Parâmetro *Size* do Tag de Comunicação que gerou a exceção
- **Elemento 6 (índice 5):** Parâmetro *Dispositivo* do Tag de Comunicação que gerou a exceção
- **Elemento 7 (índice 6):** Parâmetro *Item* do Tag de Comunicação que gerou a exceção

Utilização do Tag Especial

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

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

```
If TagExc == 0
    Return
EndIf
```

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

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

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


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Leitura de Memória de Massa

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

Leitura por Callbacks

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

Funções Especiais para Leitura de Memória de Massa

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

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

Leitura de Buffer de Eventos em Controladores GE PAC RX7

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

Tags Reportados por Eventos

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

Configuração por Strings

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

Configuração Numérica

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

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

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

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

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

Campos do Tag Bloco

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

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

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

Tags Reportados por Eventos por Ponto

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

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

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

Configuração por Strings

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

Configuração Numérica

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

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

Tags de Tempo Real

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

Configuração por Strings

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

Configuração Numérica

- **N1:** *Slave ID*
- **N2:** Código da operação
- **N3:** 100 + Índice do ponto
- **N4:** Endereço base da pilha de eventos no CLP
- **Valor:** Status do ponto

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

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

Tags:

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

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

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

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

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

Configuração por Strings

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

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

Configuração Numérica

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

- **B1:** 1000 + Endereço do escravo (relé) na rede (entre 1 e 247)
- **B2:** Código da operação configurada com a função de leitura especial SP SOE
- **B3:** 0 (zero)
- **B4:** Zona ou tabela de eventos (1 ou 2)

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

Significados dos Elementos de Bloco (tipo de dados SP_events)

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

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

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

Tag de Leitura de Evento Único

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

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

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

do tempo de varredura pode evitar a perda de eventos.

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

Deve ser configurado com os seguintes parâmetros:

Configuração por Strings

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

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

Configuração Numérica

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

Opções para o endereço do evento (Events Zone Offset)

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

Exemplos:

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

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

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

Opções disponíveis para Valor

VALOR	SIGNIFICADO
00	Borda de descida
01	Borda de subida

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

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

Algoritmo de Leitura de SOE Genérico da Elipse Software

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

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

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

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

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

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

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

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

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

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

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

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

Tabela de Eventos

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

Disposição de eventos do buffer circular

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

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

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

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

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

- **Status da tabela:** Deve ser mantido pelo CLP, indicando o número de eventos disponíveis para a leitura no *buffer* circular. Deve ser atualizado pelo dispositivo sempre que novos eventos forem adicionados ao *buffer* circular, ou após a conclusão da coleta de eventos pela aplicação, o que pode ser detectado pelo **Status da aquisição**.

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

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

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

Endereços de registradores

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

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

Procedimento de Aquisição no CLP

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

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

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

Se o número de eventos disponíveis for maior que o **máximo permitido em um único frame de comunicação** do protocolo, o Driver realiza múltiplas leituras em bloco, atualizando após a conclusão do

processo o valor do Status da Aquisição com o número total de eventos lidos.

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

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

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

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

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

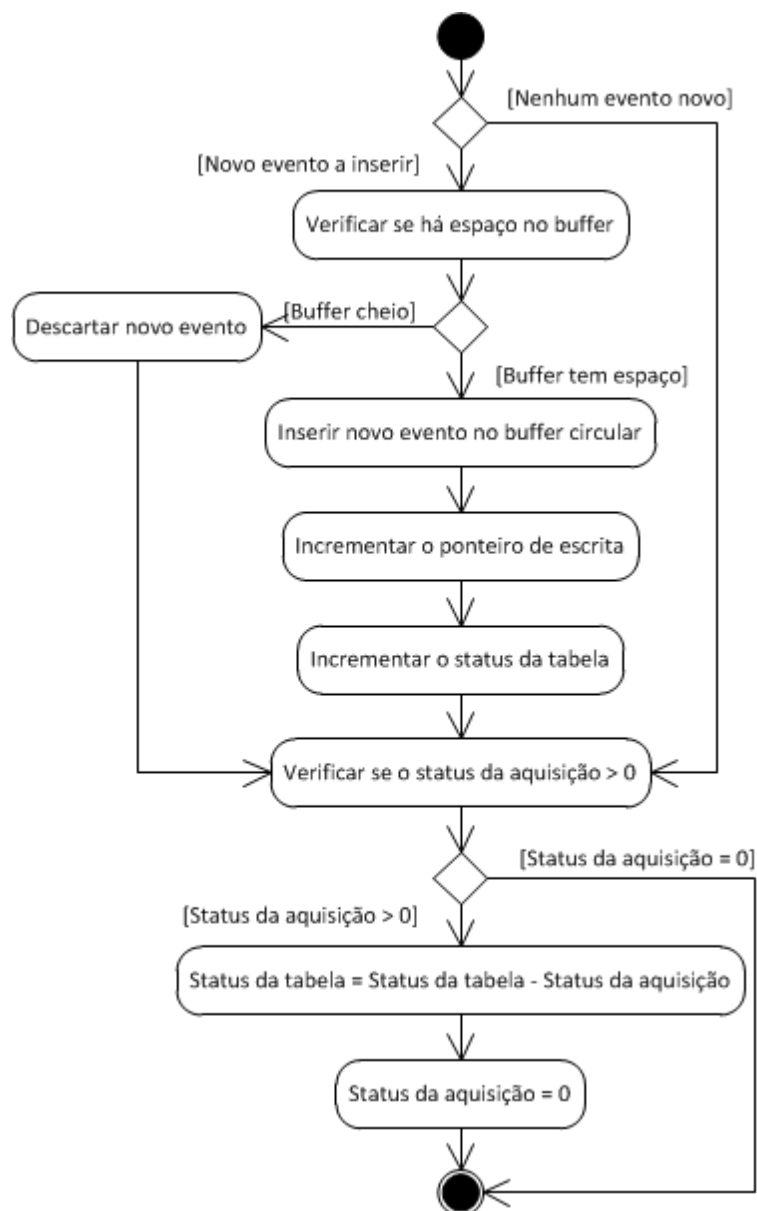


Diagrama de Atividade UML (CLP)

Estampa de Tempo

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

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

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

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

No tópico seguinte, **Procedimento de Aquisição na Aplicação**, descreve-se como configurar o supervisor

para a coleta dos eventos acumulados no CLP ou dispositivo escravo programável.

Procedimento de Aquisição na Aplicação

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

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

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

Configuração por Strings

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

Onde:

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

Exemplo:

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

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

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

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

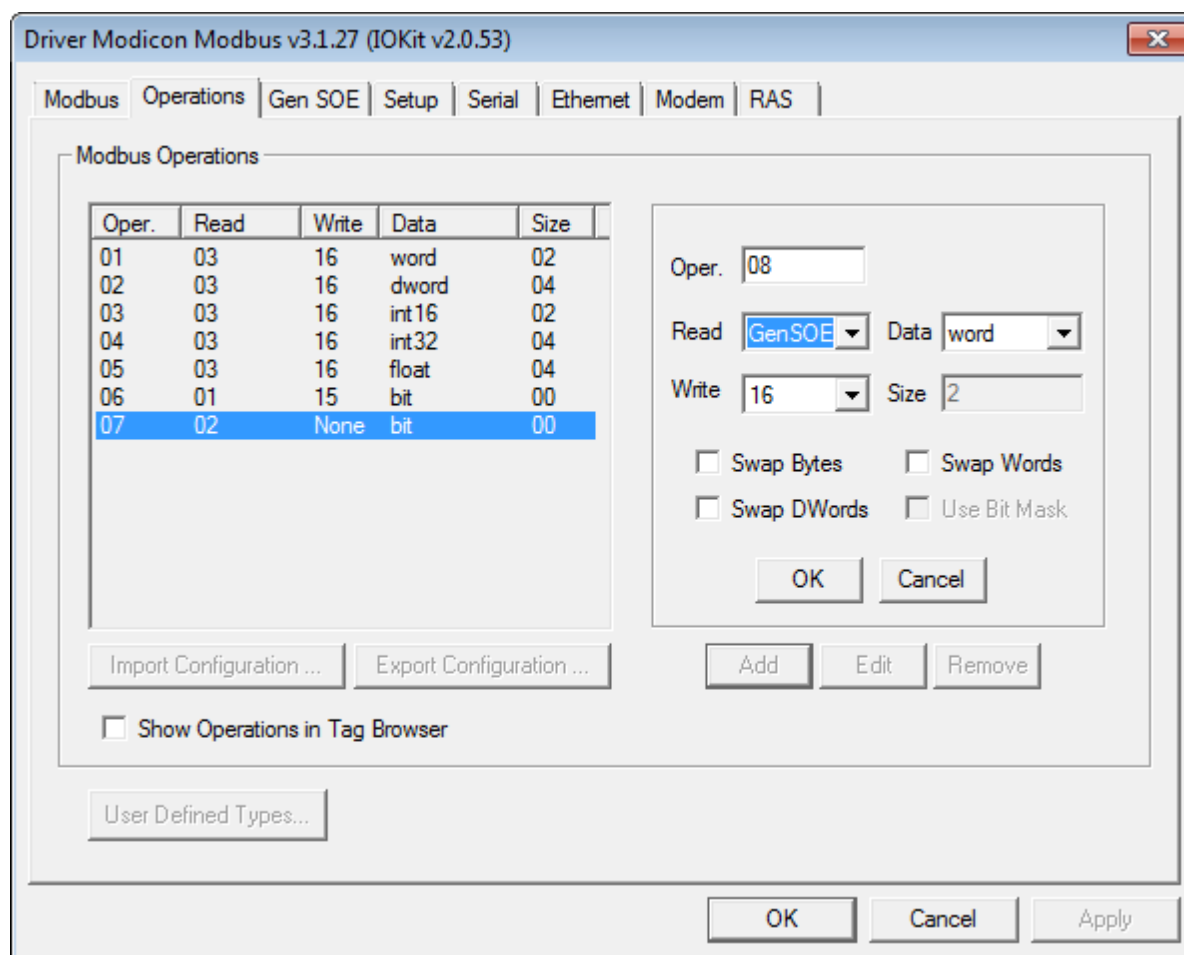
Note que, conforme já explicado, o valor do *timestamp*, embora ocupe registros no CLP, não necessita de Elementos de Tag Bloco, pois seu valor é retornado no campo *timestamp* do Tag.

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

Configuração Numérica (Parâmetros N e B)

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

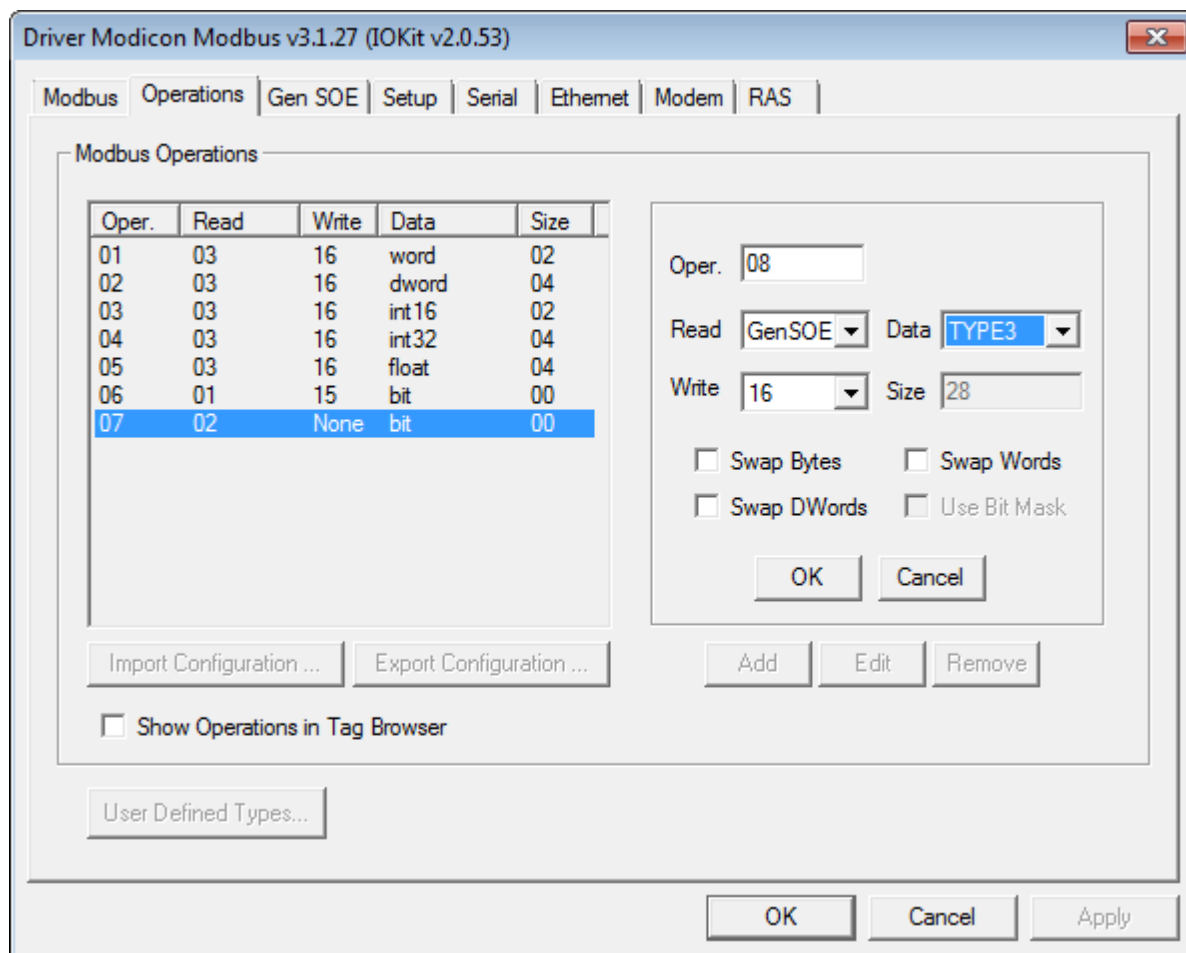
A figura a seguir mostra um exemplo de adição de operação utilizando a função especial **GenSOE** com o tipo de dados **Word**.



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 ela for suportada pelo equipamento.

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



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

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


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

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

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

- **Size:** 6

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

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

Utilização

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

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

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

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

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

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

Otimização e Compatibilidade

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

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

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

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

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

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

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

Configuração por Strings

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

Onde:

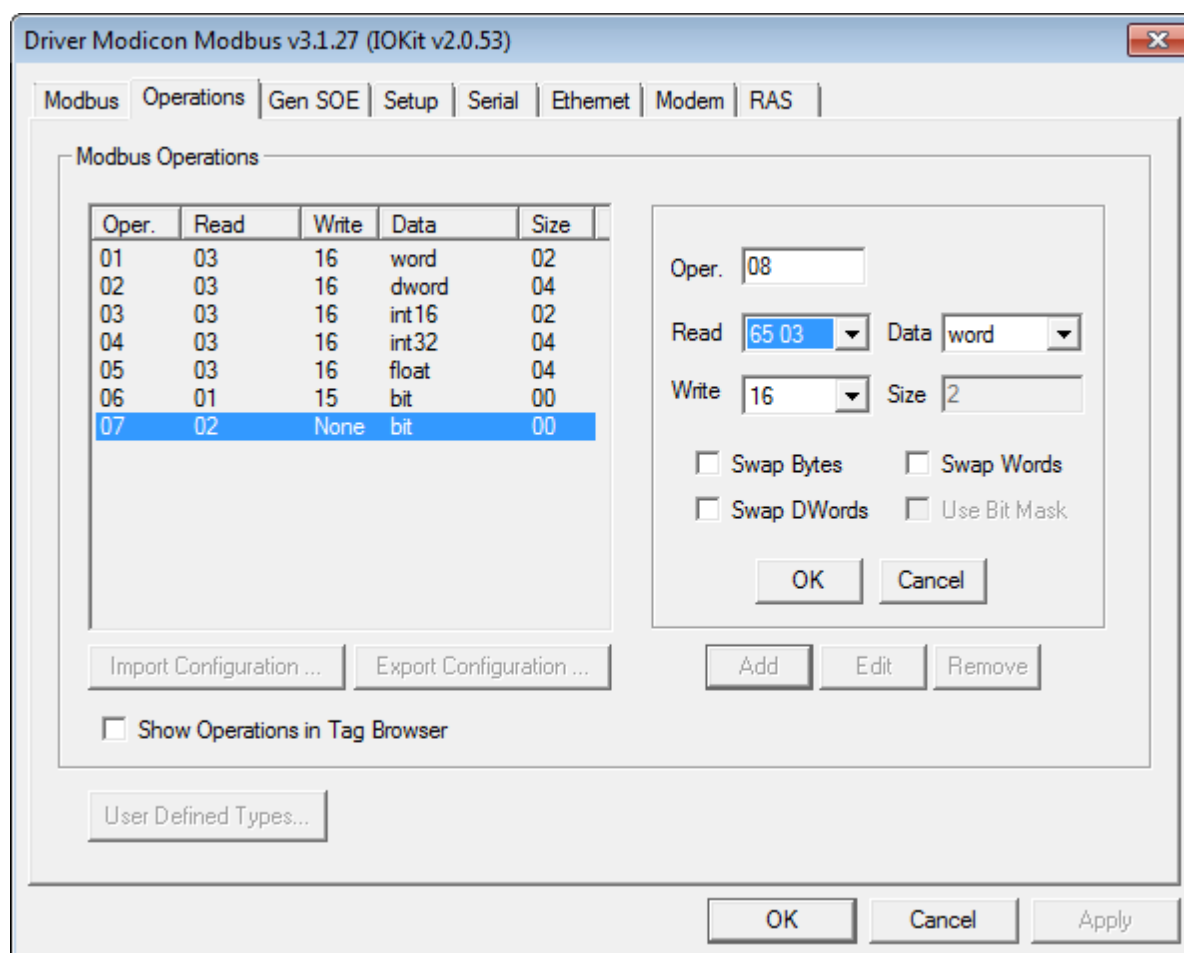
- **Endereço:** Endereço do registrador Modbus a ler
- **Tipo:** Tipo de dados. Se omitido, assume o padrão **Word**. Para mais informações, consulte o tópico **Configuração por Strings**
- **Tam. do tipo:** Usado somente em tipos de dados de tamanho variável. Para mais informações, consulte o tópico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

Configuração Numérica (Parâmetros N ou B)

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

Para a configuração de Tags numericamente, é preciso antes adicionar uma nova operação com a função **65**

03 na aba **Operations** da janela de configurações do Driver, como mostrado na figura a seguir.



Criação de uma operação com a função especial 65 03

Apêndice

Este apêndice contém os seguintes tópicos:

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

Dicas de Otimização

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

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

- Use Superblocos sempre que possível, dando preferência à criação de Tags simples (Tags PLC no Elipse SCADA) ao invés de Tags Bloco (veja o tópico **Leitura por Superblocos**).

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

Dicas de Configuração do Driver para o Elipse SCADA

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

Dicas para a Configuração ou Programação do Equipamento

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

Dúvidas Mais Frequentes

Particularidades conhecidas do equipamento Twido da empresa Schneider

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

Particularidades conhecidas do equipamento MPC 6006 da empresa Atos - Schneider

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

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

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

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

- Existe mais de um equipamento na rede serial, cada um com endereço único. Como fazer para comunicar com cada um deles?
- **Resposta:** Deve-se ter cuidado apenas com o *Slave Id* de cada **Tag de Comunicação**, pois é neste campo que deve-se indicar com qual equipamento deseja-se comunicar. Em redes seriais RS485, todos os equipamentos escutam ao mesmo tempo as requisições do Driver (há um barramento

único), porém apenas o que possuir o *Slave Id* correspondente responde à requisição (não podem haver múltiplos equipamentos com o mesmo *Id*). Na configuração por **Strings**, este valor pode ser fornecido no campo **Dispositivo**, ou no início do campo **Item** (consulte mais informações no tópico **Configuração por Strings**). No caso da **configuração numérica**, este valor é fornecido nos parâmetros *N1/B1* de cada Tag. Pode-se usar as mesmas **operações** para Tags de diversos equipamentos. Uma boa referência para maiores informações sobre a instalação e manutenção de redes seriais é o livro *Serial Port Complete*, de Jan Axelson.

- Existe mais de uma porta serial no computador. Como configurar o Driver para comunicar com os equipamentos que estão ligados em cada uma das portas?
 - **Resposta:** Neste caso, como existe mais de um meio físico diferente (Serial 1, Serial 2, etc.), são necessários tantos Drivers de Comunicação quantas portas existirem. As configurações dos **Tags** do Driver podem ser as mesmas para todos os objetos (instâncias) do Driver. A única diferença é que um Driver deve ser configurado para comunicar pela porta Serial 1, outro Driver configurado para comunicar pela porta Serial 2, e assim por diante. As configurações da porta a ser usada são realizadas na aba **Serial** da janela de configurações do Driver (veja o tópico **Propriedades**).

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

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

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

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

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

ou **ModbusASC**). O endereço IP deve ser configurado na aba **Ethernet** do IOKit, na **janela de configurações** do Driver.

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

Software Simulador Modbus

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

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

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

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

- O equipamento está comunicando diretamente na porta serial RS232 do computador. Como devo configurar os controles **RTS** e **DTR**?
- **Resposta:** Deve-se consultar a documentação do equipamento ou o suporte do fabricante para saber a configuração correta.
- O equipamento está comunicando através de um conversor RS232-RS485 ligado à porta serial RS232 do computador. Como configurar os controles **RTS** e **DTR**?
- **Resposta:** Na comunicação com equipamentos usando conversores RS232-RS485, tais configurações dependem do conversor. O equipamento (*Slave*) não tem influência, já que estes sinais só existem no lado serial RS232, não tendo equivalentes no meio serial RS485. O controle **RTS** é geralmente utilizado em conversores mais antigos para o chaveamento entre os modos de transmissão e recepção (o RS485 é *half-duplex*), devendo nestes casos em geral ser configurado no modo **Toggle** (existem alguns raros equipamentos que exigem outras configurações). Na maioria dos conversores mais recentes, entretanto, o chaveamento entre transmissão e recepção é automático, e estes

sinais em geral não são mais usados, podendo ser ignorados. Em caso de dúvidas, consulte o manual do conversor ou o suporte do fabricante.

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

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

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

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

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

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

- A aplicação no E3 ou no Elipse Power está tentando ler registros ou bits do CLP, mas sempre ocorrem erros.
 - **Resposta:** Os equipamentos desta série não permitem o uso de Superblocos do E3 ou do Elipse Power por dois motivos:
 - Existem descontinuidades no mapa de endereços de registradores dos equipamentos, com intervalos de endereços não definidos.
 - O tamanho máximo do **PDU** é diferente do estabelecido pelo padrão do protocolo, sendo definido como um tamanho que suporte 96 **Words** ou **Bits**. Uma vez que o protocolo agrupa oito bits em cada byte de dados, isto resulta em tamanhos máximos de **PDU** diferentes para as funções de leitura de **Bits** e **Words**, o que impossibilita o uso da customização do tamanho máximo do **PDU** permitida pelo Driver, que não permite configurar limites diferentes para cada

função do protocolo.

• **Solução:** Siga estes passos:

1. Desabilite a leitura por Superblocos, configurando a propriedade **EnableReadGrouping** do Driver para Falso.
2. Dê preferência à definição de Tags Bloco, agrupando o maior número possível de variáveis no menor número de Blocos, respeitando o limite do equipamento de 96 **Words** ou 96 **Bits** em cada Bloco (para mais informações, leia a seção sobre **Agrupamento Manual** no tópico **Leitura por Superblocos**).

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

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

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

Lista de Equipamentos que Comunicam com o Modbus

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

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

Equipamentos que comunicam com o Modbus

FABRICANTE	EQUIPAMENTO
ABB	<ul style="list-style-type: none">• ETE30• MGE 144• KT97• KT98

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

FABRICANTE	EQUIPAMENTO
LG / LSIS	<ul style="list-style-type: none"> • LG Master K120 S • LG XGB - XBM
Moeller	<ul style="list-style-type: none"> • 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 • XV400 - Porta Serial 232 e Porta Ethernet
Novus	<ul style="list-style-type: none"> • N1100 • N1500 • N2000 • N3000
Schneider	<ul style="list-style-type: none"> • Twido • A340 • CLP M340 • Série Premium • Conversores de frequência e <i>soft starter</i> • Disjuntores de MT e BT • Relés de proteção de BT e MT • Relés SEPAM Séries 20, 40 e 80
Telemecanique	<ul style="list-style-type: none"> • Controladores Zelio Logic com final "BD"
Unitronics	<ul style="list-style-type: none"> • V120
Weg	<ul style="list-style-type: none"> • TP 03 • Clic 02
Yaskawa	<ul style="list-style-type: none"> • V1000

Lista de Exceções Padrão do Protocolo

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

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

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

Códigos de exceção padronizados pelo protocolo Modbus

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

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

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

Limite Máximo para o Tamanho dos Blocos Suportado pelo Protocolo

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

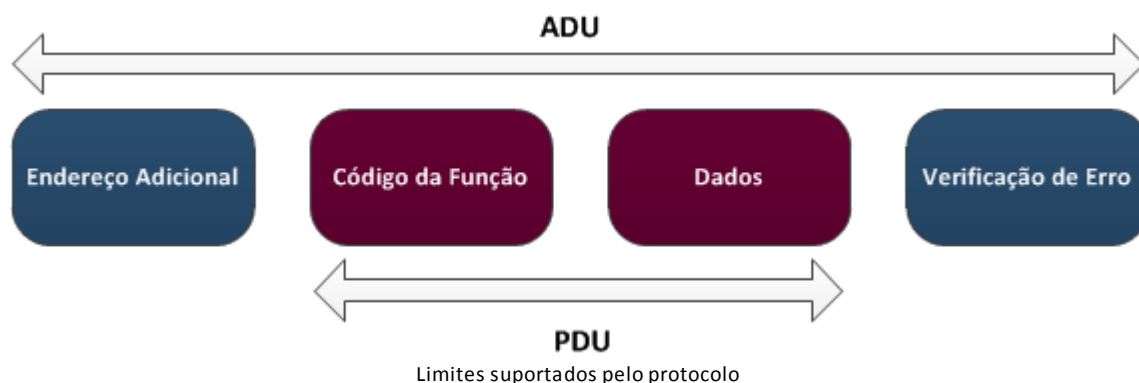
Note que, devido aos recursos de **Leitura por Superblocos** e **Partição Automática de Blocos**, presentes na versão atual do Driver, dificilmente o usuário necessita levar em conta estes limites em uma aplicação, uma vez que o Driver já realiza automaticamente as otimizações necessárias no momento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

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

Limites Suportados pelo Protocolo

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

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



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

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

Limites de Elementos de Bloco

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

FUNÇÃO MODBUS	DESCRIÇÃO	LIMITE
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)

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

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

Codificação BCD

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

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

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

Algarismos decimais em codificação BCD

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

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

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

Exemplo

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

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

- **Dígito 1:** 8
- **Dígito 2:** 4

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

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

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

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

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

Algarismos decimais em codificação BCD Comprimido

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

Histórico de Revisões do Driver

VERSÃO	DATA	AUTOR	COMENTÁRIOS
3.1.29	22/02/2016	A. Quites	<ul style="list-style-type: none">• Adicionada a configuração de Tags por Strings (<i>Case 19119</i>).• Adicionado novo Tag Browser com Tags configurados por Strings (<i>Case 20460</i>).• Melhorias na verificação de erros na configuração de tipos do usuário (<i>Case 20415</i>).• Corrigido um problema na leitura e escrita de tipos de dados BCD de oito dígitos (<i>Case 19733</i>).• Corrigida uma vulnerabilidade na qual o Driver permitia ao usuário configurar estruturas com nomes de tipos de dados nativos, confundindo o usuário (<i>Case 19816</i>).• Corrigido um erro na leitura de tipos de dados UTC32 em Bloco (<i>Case 19819</i>).• Corrigido um erro ao escrever tipos de dados Double em Bloco, ou seja, com a escrita de mais de um Elemento ao mesmo tempo (<i>Case 20053</i>).• Corrigido um erro na geração de arquivos INI de operações em formato antigo da versão 1.0 do Driver, recurso usado para retrocompatibilidade (<i>Case 20203</i>).• Corrigido um erro pelo qual a configuração equivocada do <i>byte order</i> para tipos de dados BCD gerava alteração de valores (<i>Case 20204</i>).

VERSÃO	DATA	AUTOR	COMENTÁRIOS
			<ul style="list-style-type: none"> • Corrigido um erro pelo qual a rotina Elipse SOE não retornava valores reportados a eventos para tipos de dados nativos, não estruturados (<i>Case 20364</i>). • Corrigido um erro na leitura de Tags de tempo real de eventos do equipamento GE PAC RX7, quando usados com <i>callbacks</i> (<i>Case 20374</i>).

VERSÃO	DATA	AUTOR	COMENTÁRIOS
3.0.11	29/05/2015	A. Quites	<ul style="list-style-type: none"> • Migração do Driver para o IOKit 2.0 (<i>Case 13891</i>). • Corrigido problema na janela de configuração de operações, em que a remoção de uma operação da tabela poderia falhar (<i>Case 14874</i>). • Corrigido erro no qual o Driver poderia apresentar comportamentos estranhos com a opção Reconnect after timeout habilitada, quando lendo Tags de leitura de SOE por <i>callbacks</i>. Este erro poderia, em situações raras, permitir que o Driver tentasse desconectar múltiplas vezes consecutivas, e então se reconectar também múltiplas vezes consecutivas (<i>Case 14775</i>). • Corrigido vazamento de <i>handles</i> no <i>download</i> de eventos do controlador GE PAC RX7 (<i>Case 16404</i>). • Corrigido erro no recurso de Use Bit Mask quando usado com Superblocos habilitados (<i>Case 18340</i>). • Corrigido erro na leitura de Strings com tamanho ímpar, em que os últimos caracteres poderiam ser truncados caso as opções de <i>swap</i> estivessem habilitadas (<i>Case 16744</i>).

VERSÃO	DATA	AUTOR	COMENTÁRIOS
2.8.17	19/10/2012	A. Quites	<ul style="list-style-type: none"> • Implementado o Elipse Modbus SOE, recurso que padroniza a leitura de SOE para a maioria dos CLPs (<i>Case 12038</i>). • Adicionados os tipos ou estruturas definidos pelo usuário, como parte da implementação do recurso de SOE Genérico (<i>Case 12038</i>). • Implementada a leitura de SOE para relés Schneider Electric séries SEPAM 20, 40 e 80 (<i>Case 12106</i>). • Adicionado o novo tipo Sp_time para a representação de estampas de tempo (<i>timestamps</i>) de relés Schneider Electric séries SEPAM (<i>Case 12106</i>). • Adicionados os tipos de data e hora GenTime, UTC64d e UTC32, que podem ser usados para a representação de estampas de tempo (<i>timestamp</i>) (<i>Case 12038</i>). • Adicionada a leitura por <i>callbacks</i> para Tags de leitura de SOE (<i>Case 12464</i>). • Adicionada opção de reconexão em caso de ocorrência de <i>timeout</i> na recepção de <i>frames</i>, quando utilizada a camada física Ethernet (<i>Case 12537</i>). • Remodelada a janela de configuração para permitir a adição das novas opções de configuração, com a criação de nova aba Operations (<i>Case 12038</i>).

VERSÃO	DATA	AUTOR	COMENTÁRIOS
			<ul style="list-style-type: none"> • Removida a opção Swap Address Delay da janela de configuração do Driver. Esta opção, tornada obsoleta pela opção Inter-frame delay do IOKit, continua disponível como configuração <i>offline</i>, mantendo compatibilidade com aplicações legadas (<i>Case 13285</i>). • Removida a opção Reverse Frame da caixa de diálogo de configuração das operações do Driver, por obsolescência. A opção continua sendo suportada em aplicações legadas (<i>Case 12443</i>). • Adicionadas informações sobre a configuração de tipos Float para controladores WEG TPW-03 (<i>Case 12546</i>). • Adicionada informação sobre a configuração do Driver para o controlador ABB 07KT97 e para todos os controladores da família ABB Advant Controller 31 series 90 (<i>Case 12667</i>). • Adicionado o CLP ATOS à lista de equipamentos que suportam o protocolo Modbus (<i>Case 13247</i>). • Adicionadas informações sobre o recurso de partição automática de blocos, disponível a partir da versão 2.00 (<i>Case 11594</i>). • Adicionado o CLP WEG Clic 02 à lista de equipamentos suportados (<i>Case 11602</i>). • Revisado o texto do tópico de introdução para deixar claro que este Driver trabalha como Mestre na rede Modbus, permitindo a comunicação com dispositivos Escravos (<i>Case 12035</i>).

VERSÃO	DATA	AUTOR	COMENTÁRIOS
			<ul style="list-style-type: none"> • Adicionado tópico sobre Superblocos (propriedade EnableReadGrouping) (<i>Case 13287</i>). • Adicionado tópico sobre particionamento automático de blocos (<i>Case 13288</i>). • Adicionado tópico com dicas de otimização (<i>Case 13287</i>). • Adicionada ao tópico Dúvidas Mais Frequentes a questão sobre a mensagem referente à recepção de valores Float não normalizados (<i>Case 13295</i>). • Atualizada a lista de equipamentos que comunicam usando o protocolo Modbus, levando em conta equipamentos registrados nos suportes atendidos mais recentemente (<i>Case 13298</i>). • Adicionado o tópico Guia de Configuração Rápida, com o passo a passo para a configuração do Driver nos casos de uso mais comuns (<i>Case 13301</i>). • Solucionado erro ao carregar arquivo de configuração no Windows CE ARM HPC2000 (<i>Case 12352</i>). • Solucionado erro na escrita de valores float_GE (<i>Case 12298</i>). • Solucionado erro no qual o último caractere de String lida, com tamanho ímpar de caracteres, poderia deixar de ser retornado à aplicação (<i>Case 12466</i>).
2.7.1	30/06/2010	A. Quites	<ul style="list-style-type: none"> • Ajustes na recepção dos dados para descartar possíveis bytes inválidos (<i>Case 11394</i>).

VERSÃO	DATA	AUTOR	COMENTÁRIOS
		C. Mello	<ul style="list-style-type: none"> • Atualizado o tópico Leitura do Código da Última Exceção (<i>Case 11233</i>). • Atualizado o tópico Dúvidas Mais Frequentes (<i>Case 11316</i>).
2.6.3	26/11/2009	A. Quites	<ul style="list-style-type: none"> • Removida a mensagem de erro falsa no arquivo de log para leitura de dados de memória com endereço zero (<i>Case 10654</i>). • Otimizada a leitura do tipo Bit via Superblocos (<i>Case 10971</i>).
		C. Mello	<ul style="list-style-type: none"> • Modificada a rotina de <i>Wait Silence</i> para ser executada em qualquer erro que ocorrer no Driver. O usuário também pode executar a rotina <i>Wait Silence</i> manualmente pelo novo Tag Especial N2 = 9001 (<i>Case 10850</i>). • Adicionada compatibilidade com a plataforma Windows CE (<i>Case 10914</i>).
2.5.12	30/06/2009	A. Quites	<ul style="list-style-type: none"> • Corrigido erro na opção Swap Address Delay (<i>Case 10425</i>). • Atualizado o valor padrão para o tamanho máximo do PDU, conforme versão mais recente do padrão Modbus (<i>Case 10274</i>). • Solucionado erro nos Tags especiais de última exceção, que poderiam não reportar algumas exceções (<i>Case 10337</i>). • Corrigido erro no retorno do ano para registros de SOE com o dia atual do controlador GE PAC RX7 (<i>Case 10382</i>). • Corrigido erro na função 20 (<i>Read File Record</i>) (<i>Case 10312</i>).

VERSÃO	DATA	AUTOR	COMENTÁRIOS
		M. Ludwig	<ul style="list-style-type: none"> • Criado novo Tag para ler eventos de SOE de pontos específicos do GE PAC RX7 (<i>Case 10370</i>).
2.4.10	17/02/2009	A. Quites	<ul style="list-style-type: none"> • Corrigido erro na leitura de tipos Bit com Superblocos (<i>Case 10100</i>). • Implementada leitura do evento GE SOE mais recente (<i>Case 10178</i>). • Criadas operações padrão para os tipos de funções mais comuns (<i>Case 9185</i>).
2.3.22	02/09/2008	C. Mello	<ul style="list-style-type: none"> • Adicionado novo tipo de dado Float_GE (<i>Case 9427</i>).

VERSÃO	DATA	AUTOR	COMENTÁRIOS
		A. Quites	<ul style="list-style-type: none"> • Corrigido erro na configuração <i>offline</i> do parâmetro <i>ModiconModbus.Modbus Mode (Case 9831)</i>. • Implementada a leitura de máscaras de bits de registradores (<i>Case 9682</i>). • Implementada a leitura de <i>buffer</i> de eventos em controladores GE PAC RX7 (<i>Case 9523</i>). • Implementada a possibilidade de configurar um limite máximo para o PDU (<i>Case 9154</i>). • Corrigido erro nos recursos de reordenamento de bytes (Swap Byte, Swap Word, Swap DWord e Rev. Frame) na leitura de tipos Word com Superblocos habilitados (<i>Case 9220</i>). • Implementada a opção Enable CMS Addressing (<i>Case 8665</i>). • Revisões e testes em funções Modbus pouco usadas (<i>Case 8730</i>). • Melhorada a consistência de dados escritos através da função Modbus 06 (Write Single Register) (<i>Case 8663</i>).
2.2	11/05/2007	A. Quites	<ul style="list-style-type: none"> • Corrigido erro na leitura de blocos de tipos não Word com Superblocos habilitados (<i>Case 8243</i>).

VERSÃO	DATA	AUTOR	COMENTÁRIOS
2.1	23/01/2007	A. Quites	<ul style="list-style-type: none"> • Implementado o recurso de Superblocos (<i>Case 6185</i>). • Melhorada a consistência de parâmetros N2/B2 de Tags de acesso a registros (<i>Case 7714</i>). • Corrigido erro na leitura de blocos de dados do tipo BCD de tamanho quatro (<i>Case 7728</i>). • Corrigido erro na leitura de blocos de Strings (<i>Case 7804</i>).
2.0	10/11/2006	A. Quites	<ul style="list-style-type: none"> • Versão original com IOKit (<i>Case 3339</i>).
1.0		R. Farina	<ul style="list-style-type: none"> • Todas as versões anteriores ao controle de revisões.

**Matriz**

Rua 24 de Outubro, 353 - 10º andar
90510-002 Porto Alegre
Fone: (+55 51) 3346-4699
Fax: (+55 51) 3222-6226
E-mail: elipse-rs@elipse.com.br

Filial PR

Av. Sete de Setembro, 4698/1705
80240-000 Curitiba - PR
Fone: (+55 41) 4062-5824
E-mail: elipse-pr@elipse.com.br

Filial RJ

Praia de Botafogo, 300/525
22250-044 Rio de Janeiro - RJ
Fone: (+55 21) 2158-1015
Fax: (+55 21) 2158-1099
E-mail: elipse-rj@elipse.com.br

Filial SP

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

Filial MG

Rua Antônio de Albuquerque, 156
7º andar Sala 705
30112-010 Belo Horizonte - MG
Fone: (+55 31) 4062-5824
E-mail: elipse-mg@elipse.com.br

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, 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)