



# Manual de Programação

## IEC 61131

Rev. D 09/2015  
Cód. Doc.: MP399048



altus

[www.altus.com.br](http://www.altus.com.br)





Nenhuma parte deste documento pode ser copiada ou reproduzida sem o consentimento prévio e por escrito da Altus Sistemas de Automação S.A., que se reserva o direito de efetuar alterações sem prévio comunicado.

Conforme o Código de Defesa do Consumidor vigente no Brasil, informamos a seguir, aos clientes que utilizam nossos produtos, aspectos relacionados com a segurança de pessoas e instalações.

Os equipamentos de automação industrial fabricados pela Altus são robustos e confiáveis devido ao rígido controle de qualidade a que são submetidos. No entanto, equipamentos eletrônicos de controle industrial (controladores programáveis, comandos numéricos, etc.) podem causar danos às máquinas ou processos por eles controlados em caso de defeito em suas partes e peças ou de erros de programação ou instalação, podendo inclusive colocar em risco vidas humanas.

O usuário deve analisar as possíveis consequências destes defeitos e providenciar instalações adicionais externas de segurança que, em caso de necessidade, sirvam para preservar a segurança do sistema, principalmente nos casos da instalação inicial e de testes.

Os equipamentos fabricados pela Altus não trazem riscos ambientais diretos, não emitindo nenhum tipo de poluente durante sua utilização. No entanto, no que se refere ao descarte dos equipamentos, é importante salientar que quaisquer componentes eletrônicos incorporados em produtos contêm materiais nocivos à natureza quando descartados de forma inadequada. Recomenda-se, portanto, que quando da inutilização deste tipo de produto, o mesmo seja encaminhado para usinas de reciclagem que deem o devido tratamento para os resíduos.

É imprescindível a leitura completa dos manuais e/ou características técnicas do produto antes da instalação ou utilização do mesmo.

A Altus garante os seus equipamentos conforme descrito nas Condições Gerais de Fornecimento, anexada às propostas comerciais.

A Altus garante que seus equipamentos funcionam de acordo com as descrições contidas explicitamente nos manuais e/ou características técnicas, não garantindo a satisfação de algum tipo particular de aplicação dos equipamentos.

A Altus desconsiderará qualquer outra garantia direta ou implícita, principalmente quando se tratar de fornecimento de terceiros.

Pedidos de informações adicionais sobre o fornecimento e/ou características dos equipamentos e serviços Altus devem ser feitos por escrito. A Altus não se responsabiliza por informações fornecidas sobre seus equipamentos sem registro formal.

## DIREITOS AUTORAIS

Nexto e MasterTool IEC XE são marcas registradas da Altus Sistemas de Automação S.A.

Windows é marca registrada da Microsoft Corporation.

# Sumário

<b>SUMÁRIO .....</b>	<b>II</b>
<b>1. INTRODUÇÃO .....</b>	<b>6</b>
<b>Documentos Relacionados a este Manual .....</b>	<b>6</b>
Considerações Gerais sobre Documentação ALTUS .....	7
Documentação de Suporte ao MasterTool IEC XE .....	7
<b>Inspeção Visual .....</b>	<b>7</b>
<b>Suporte Técnico .....</b>	<b>7</b>
<b>Mensagens de Advertência Utilizadas neste Manual .....</b>	<b>8</b>
<b>2. CONCEITOS E COMPONENTES BÁSICOS .....</b>	<b>9</b>
<b>Introdução .....</b>	<b>9</b>
<b>Conceitos Básicos .....</b>	<b>9</b>
<b>Funcionalidades Avançadas .....</b>	<b>9</b>
Orientação a Objetos na Programação e na Estrutura do Projeto .....	9
Tipos de Dados Especiais .....	10
Operadores e Variáveis Especiais .....	10
Conceito de Gerenciamento de Usuário e Direitos de Acesso .....	10
Características em Editores .....	10
Versões de Bibliotecas .....	10
Funcionalidades Adicionais .....	10
<b>Perfis .....</b>	<b>11</b>
<b>Projeto .....</b>	<b>11</b>
<b>Dispositivos .....</b>	<b>11</b>
<b>Aplicação .....</b>	<b>12</b>
<b>Task Configuration .....</b>	<b>12</b>
Notas Importantes para Sistemas Multitarefa .....	12
<b>Comunicação .....</b>	<b>13</b>
<b>Geração de Código e Alteração Online .....</b>	<b>13</b>
Geração de Código e Informações de Compilação .....	13
Alterações Online .....	13
Aplicação de Inicialização (Projeto de Inicialização) .....	13
Método de Envio/Login de Projetos Sem Diferença de Projetos .....	14
<b>Monitoração .....</b>	<b>14</b>
<b>Depuração .....</b>	<b>14</b>
<b>Linguagens de Programação Suportadas .....</b>	<b>14</b>
<b>Unidades de Organização de Programas .....</b>	<b>15</b>
POU .....	15
Programa .....	17
Função .....	18
Bloco Funcional .....	20
Unidades de Tipo de Dados .....	24
Método .....	25
Propriedade .....	26
Ação .....	27
Função Externa, Bloco Funcional, Método .....	28
Lista de Variáveis Globais - GVL .....	28
Variáveis Persistentes .....	29
Arquivo Externo .....	29
POUs para Verificações Implícitas .....	30

<b>Gerenciamento de Bibliotecas .....</b>	<b>30</b>
Instalação e Inclusão no Projeto .....	31
Bibliotecas Referenciadas .....	31
Versões de Biblioteca .....	32
Acesso Único aos Módulos da Biblioteca ou às Variáveis .....	32
Criando Bibliotecas, Codificação, Documentação .....	33
<b>3. COMANDOS DO MENU .....</b>	<b>35</b>
<b>Library Manager .....</b>	<b>35</b>
Comandos do Gerenciador de Bibliotecas .....	35
Acrescentar Biblioteca .....	35
Propriedades .....	38
Tentar Recarregar a Biblioteca .....	39
<b>4. REFERÊNCIA DE PROGRAMAÇÃO .....</b>	<b>40</b>
<b>Declaração.....</b>	<b>40</b>
Declaração de Variáveis.....	40
Recomendações na Nomeação de Identificadores .....	41
Inicialização de Variáveis .....	45
Expressões Arbitrárias para Inicialização de Variáveis .....	45
Editor de Declaração.....	45
Diálogo Autodeclarar.....	46
Modo Atalho .....	46
Declaração AT.....	46
Palavras-chave.....	47
Variáveis Locais VAR .....	48
Variáveis de Entrada - VAR_INPUT.....	48
Variáveis de Saída - VAR_OUTPUT .....	48
Variáveis de Entrada e Saída - VAR_IN_OUT .....	48
Variáveis Globais - VAR_GLOBAL .....	49
Variáveis Temporárias - VAR_TEMP.....	49
Variáveis Estáticas - VAR-STAT.....	49
Variáveis Externas – VAR_EXTERNAL.....	49
Atributo Palavras-chave para Tipos de Variáveis.....	50
Variáveis Remanentes .....	50
Constantes .....	52
Configuração de Variáveis – VAR_CONFIG .....	53
Declaração e Inicialização de Tipos de Dados Definidos pelo Usuário.....	54
Métodos FB_Init e FB_Reinit .....	54
FB_Exit .....	55
Instruções de Pragma .....	56
Funcionalidade Listar Componentes .....	71
<b>Mapeamentos de E/S.....</b>	<b>71</b>
Geral .....	71
Canais.....	72
<b>Tipos de Dados.....</b>	<b>73</b>
Tipos de Dados Padrão .....	73
Extensões da Norma IEC 1131-3 .....	75
Tipos de Dados Definidos pelo Usuário .....	77
<b>Operadores.....</b>	<b>84</b>
Operadores IEC e Funções Adicionais à Norma .....	84
Operadores Aritméticos .....	84
Operadores de Bitstring .....	89
Operadores de Deslocamento de Bits .....	91

Operadores de Seleção .....	94
Operadores de Comparação.....	97
Operadores de Endereço .....	100
Operador de Chamada.....	101
Funções de Conversão de Tipo.....	101
Funções Numéricas.....	109
Operadores Adicionais à Norma IEC.....	114
<b>Operandos .....</b>	<b>115</b>
Constantes .....	115
Variáveis .....	118
Endereços .....	120
Funções .....	122
<b>5. EDITORES DAS LINGUAGENS DE PROGRAMAÇÃO .....</b>	<b>123</b>
<b>Editor CFC .....</b>	<b>123</b>
Linguagem Gráfico Funcional Contínuo - CFC .....	123
Posições do Cursor no CFC.....	124
Elementos CFC / Caixa de Ferramentas .....	125
Inserir e Organizar Elementos .....	128
Editor CFC no Modo Online .....	130
<b>Editor SFC .....</b>	<b>131</b>
Seqüenciamento Gráfico de Funções - SFC.....	132
Posições do Cursor no SFC.....	133
Trabalhando no Editor SFC.....	134
Propriedades do Elemento SFC.....	135
Elementos SFC / Caixa de Ferramentas.....	136
Qualificador.....	144
Variáveis Implícitas - Memórias SFC.....	145
Seqüência de Processamento no SFC .....	150
Editor SFC no Modo Online .....	151
<b>Texto Estruturado (ST) / Texto Estruturado Estendido (ExST) .....</b>	<b>152</b>
Expressões.....	152
Instruções .....	154
<b>Editor ST.....</b>	<b>160</b>
Editor ST no Modo Online.....	160
<b>Editor FBD/LD/IL .....</b>	<b>163</b>
Diagrama de Blocos Funcionais - FBD .....	163
Diagrama Ladder - LD.....	164
Lista de Instruções - IL .....	164
Trabalhando na Visualização dos Editores FBD e LD .....	167
Trabalhando na Visualização do Editor IL.....	169
Posições do Cursor em FBD, LD e IL .....	173
Menu FBD/LD/IL.....	175
Elementos .....	176
<b>6. BIBLIOTECAS.....</b>	<b>188</b>
<b>A Biblioteca Standard.library .....</b>	<b>188</b>
Funções de String .....	188
Blocos Funcionais Biestáveis .....	193
Disparador .....	194
Contador.....	195
Temporizador .....	198
<b>A Biblioteca UTIL.library .....</b>	<b>202</b>
Conversão BCD.....	202

---

Funções BIT/BYTE .....	203
Função Matemática Auxiliar .....	204
Controllers.....	208
Geradores de Sinal.....	211
Manipuladores de Função .....	214
Processamento de Valores Analógicos .....	216
<b>A Biblioteca NextoPID.library .....</b>	<b>218</b>
PID.....	218
PID_REAL.....	222
PID_INT.....	224
<b>LibRecipeHandler.....</b>	<b>237</b>
WriteRecipe.....	237
<b>7. GLOSSÁRIO .....</b>	<b>239</b>

# 1. Introdução

A Série Nexto é uma poderosa e completa série de Controladores Programáveis (CP) com características exclusivas e inovadoras. Devido a sua flexibilidade, design inteligente, recursos de diagnósticos avançados e arquitetura modular, a Série Nexto pode ser usada para controle de sistemas em aplicações de médio e grande porte ou em máquinas com requisitos de alto desempenho.

O MasterTool IEC XE é uma ferramenta completa para programação, depuração, configuração e simulação das aplicações do usuário. O software é baseado no conceito de ferramenta integrada, provendo flexibilidade e facilidade de uso permitindo aos usuários a programação em seis linguagens definidas pela norma IEC 61131-3: Texto Estruturado (ST), Sequenciamento Gráfico de Funções (SFC), Diagrama de Blocos Funcionais (FBD), Diagrama Ladder (LD), Lista de Instruções (IL) e Gráfico Contínuo de Funções (CFC). O MasterTool IEC XE permite o uso de diferentes linguagens na mesma aplicação, fornecendo ao usuário uma poderosa maneira de organizar a sua aplicação além de reutilizar código desenvolvido em aplicações anteriores.

Este produto oferece características para todas as etapas de desenvolvimento de um sistema de automação, começando por uma análise gráfica da topologia da arquitetura, passando por um ambiente de programação com suporte às linguagens da norma IEC 61131-3, com uma ferramenta de simulação realística, onde o usuário pode verificar o comportamento da aplicação antes de executá-la em um sistema real, e, finalmente, provê uma completa interface para visualização de diagnósticos e status.

O MasterTool IEC XE também oferece dois diferentes mecanismos de proteção da aplicação e características de segurança: Proteção da Propriedade Intelectual e Login Seguro no CP. A Proteção da Propriedade Intelectual tem por objetivo proteger a propriedade intelectual do usuário, permitindo a ele proteger todo o projeto ou arquivos específicos dentro do projeto através da definição uma senha de acesso. Isso significa que estes arquivos estarão disponíveis (para operação de leitura e escrita) apenas depois de desbloqueados com a senha correta. Já o Login Seguro no CP provê uma maneira de proteger a aplicação do usuário de qualquer acesso não autorizado. Habilitando esta característica, a UCP da Série Nexto irá solicitar uma senha de usuário antes de executar quaisquer comandos entre MasterTool IEC XE e a UCP, como parar e programar a aplicação ou forçar pontos de saída em um módulo.

O MasterTool IEC XE torna o uso de interfaces para redes de campo uma prática tão simples como nunca visto anteriormente. O usuário não precisa um software especial para configurar a rede de campo porque o MasterTool IEC XE atende este requisito através de uma única ferramenta reduzindo tempo de desenvolvimento e simplificando a aplicação.

Outras características importantes também estão disponíveis para aumentar a produtividade do usuário como: Módulo de impressão que consiste de um relatório com os parâmetros específicos dos módulos e as configurações da aplicação; Impressão de lógicas que consiste de um relatório com todo o código da aplicação; Verificação de Projeto que auxilia o usuário a verificar diferentes condições durante a programação como: sintaxe do programa, consumo de corrente da fonte de alimentação, regras de posicionamento dos módulos da Série Nexto, parametrização e configuração de módulos; Depuração em tempo real que provê uma maneira de verificar a funcionalidade da aplicação passo-a-passo, verificar o conteúdo de variáveis ou ainda adicionar e remover breakpoints durante a programação da UCP da Série Nexto.

## Documentos Relacionados a este Manual

Para obter informações adicionais sobre o MasterTool IEC XE, podem ser consultados outros documentos específicos além deste. Estes documentos encontram-se disponíveis na sua última revisão em [www.altus.com.br](http://www.altus.com.br).



## Considerações Gerais sobre Documentação ALTUS

Cada produto possui um documento denominado Característica Técnica (CT), onde se encontram as características do produto em questão. Adicionalmente o produto pode possuir Manuais de Utilização (os códigos dos manuais, se aplicáveis, são sempre citados na CT).

## Documentação de Suporte ao MasterTool IEC XE

Aconselham-se os seguintes documentos como fontes de informação adicional:

Código do documento	Descrição	Idioma
CE114000	Nexto Series – Features and Configuration	Inglês
CT114000	Série Nexto – Características e Configurações	Português
CS114000	Serie Nexto – Especificaciones y Configuraciones	Espanhol
CE114100	CPUs Nexto Series – Features and Configuration	Inglês
CT114100	UCPs Série Nexto – Características e Configurações	Português
CS114100	UCPs Serie Nexto – Especificaciones y Configuraciones	Espanhol
CE103705	MasterTool IEC XE – Features and Configuration	Inglês
CT103705	MasterTool IEC XE – Características e Configurações	Português
CS103705	MasterTool IEC XE – Especificaciones y Configuraciones	Espanhol
MU214600	Nexto Series User Manual	Inglês
MU214000	Manual de Utilização Série Nexto	Português
MU214300	Manual Del Usuario Serie Nexto	Espanhol
MU214605	Nexto Séries CPUs User Manual	Inglês
MU214100	Manual de Utilização UCPs Série Nexto	Português
MU214305	Manual del Usuario UCPs Serie Nexto	Espanhol
MU299609	MasterTool IEC XE User Manual	Inglês
MU299048	Manual de Utilização MasterTool IEC XE	Português
MU299800	Manual del Usuario MasterTool IEC XE	Espanhol
MU399609	IEC 61131 Programming Manual	Inglês
MU399048	Manual de Programação IEC 61131	Português
MU399800	Manual de Programación IEC 61131	Espanhol

**Tabela 1-1. Documentação de Suporte**

## Inspeção Visual

Antes de proceder à instalação, é recomendável fazer uma inspeção visual cuidadosa dos equipamentos, verificando se não há danos causados pelo transporte. Verifique se todos os componentes de seu pedido estão em perfeito estado. Em caso de defeitos, informe a companhia transportadora e o representante ou distribuidor Altus mais próximo.

### **CUIDADO:**

**Antes de retirar os módulos da embalagem, é importante descarregar eventuais potenciais estáticos acumulados no corpo. Para isso, toque (com as mãos nuas) em qualquer superfície metálica aterrada antes de manipular os módulos. Tal procedimento garante que os níveis de eletricidade estática suportados pelo módulo não serão ultrapassados.**

É importante registrar o número de série de cada equipamento recebido, bem como as revisões de software, caso existentes. Essas informações serão necessárias caso seja preciso contatar o Suporte Técnico da Altus.

## Suporte Técnico

Para entrar em contato com o Suporte Técnico da Altus em São Leopoldo, RS, ligue para +55-51-3589-9500. Para conhecer os centros de Suporte Técnico da Altus existentes em outras localidades, consulte nosso site ([www.altus.com.br](http://www.altus.com.br)) ou envie um e-mail para [altus@altus.com.br](mailto:altus@altus.com.br).

Se o equipamento já estiver instalado tenha em mãos as seguintes informações ao solicitar assistência:

- os modelos dos equipamentos utilizados e a configuração do sistema instalado
- o número de série da UCP
- a revisão do equipamento e a versão do software executivo, constantes na etiqueta afixada na lateral do produto
- as informações sobre o modo de operação da UCP, obtidas através do programador MasterTool
- o conteúdo do programa aplicativo (módulos), obtido através do programador MasterTool
- a versão do programador utilizado

### Mensagens de Advertência Utilizadas neste Manual

Neste manual, as mensagens de advertência apresentarão os seguintes formatos e significados:

**PERIGO:**

Relatam causas potenciais, que se não observadas, levam a danos à integridade física e saúde, patrimônio, meio ambiente e perda da produção.

**CUIDADO:**

Relatam detalhes de configuração, aplicação e instalação que *devem* ser seguidos para evitar condições que possam levar a falha do sistema e suas consequências relacionadas.

**ATENÇÃO:**

Indicam detalhes importantes de configuração, aplicação ou instalação para obtenção da máxima performance operacional do sistema.

## 2. Conceitos e Componentes Básicos

### Introdução

O MasterTool IEC XE é um software programador de CP independente de dispositivo. A sua compatibilidade com a IEC 61131-3 permite suporte a todas as linguagens de programação definidas na norma.

### Conceitos Básicos

Considere os seguintes conceitos básicos que caracterizam a programação via MasterTool IEC XE:

- **Orientação a objetos:** a orientação a objetos é contemplada pela disponibilidade das características e dos elementos de programação associados, pelo tratamento da estrutura e pela forma como o projeto é organizado.
- **Estrutura do programador baseada em componentes:** a funcionalidade disponível na interface do usuário (editores, menus) depende dos componentes utilizados. Existem componentes essenciais e opcionais.
- **A organização do projeto é também determinada pela orientação ao objeto:** um projeto do MasterTool IEC XE contém um programa de CP composto de vários objetos de programação e também da definição dos "recursos" necessários para executar as instâncias do programa (aplicação) nos sistemas dos dispositivos definidos (dispositivos, CPs). Assim sendo, existem dois tipos de objetos em um projeto:
  - **Objetos de programação:** os objetos de programação (POUs) que podem ser instanciados no projeto (em todas as aplicações definidas no mesmo) devem ser gerenciados na janela das POU's. Alguns exemplos destes objetos de programação são: programas, funções, blocos funcionais, métodos, ações, definições de tipos de dados, entre outros. O instanciamento é feito ao chamar uma POU de programa através de uma tarefa atribuída à aplicação. Os objetos de programação gerenciados na janela de dispositivos (atribuídos diretamente a uma aplicação) não podem ser instanciados por outra aplicação inserida abaixo.
  - **Objetos de recurso:** estes são objetos de dispositivos, aplicações, configurações de tarefas e são gerenciados na árvore de dispositivos ou no editor gráfico, dependendo do tipo do dispositivo. Ao inserir objetos, o hardware a ser controlado deve ser mapeado de acordo com determinadas regras.
- **Geração de código:** a geração de código é feita através de compiladores integrados e otimização do código de máquina versus tempos de execução.
- **Transferência de dados entre o MasterTool IEC XE e o dispositivo (controlador):** esta operação é feita via gateway (componente) e um sistema de execução.
- **Interface padrão e profissional:** configurações pré-definidas oferecem a possibilidade de escolher entre uma interface de usuário "padrão" (seleção reduzida de configurações com menor complexidade) ou um ambiente "profissional", o qual suporta todas as configurações. A escolha entre estas opções é feita quando o programador é inicializado após a primeira instalação no sistema, porém é possível alterar esta opção posteriormente, assim como se pode também adotar uma customização definida pelo usuário. Para conhecer mais detalhes sobre as diferenças específicas entre as versões padrão e profissional, consulte Características no Manual de Utilização MasterTool IEC XE – MU299048.

### Funcionalidades Avançadas

A seguir, são apresentadas as funcionalidades avançadas disponíveis no MasterTool IEC XE.

#### Orientação a Objetos na Programação e na Estrutura do Projeto

Extensões para blocos funcionais: Propriedades, Métodos, Herança, Invocação de Método.

Aplicações vinculadas a dispositivos como instâncias de objetos de programação independentes.

### Tipos de Dados Especiais

- UNION
- LTIME
- Referências
- Enumerações: tipos de dados básicos podem ser especificados
- DI: DINT := DINT#16#FFFFFFFF

### Operadores e Variáveis Especiais

- Operadores de escopo: contextos estendidos
- Ponteiros de função: substituindo o operador INSTANCE\_OF
- Método Init: substituindo o operador INI
- Método Exit
- Variáveis de saída em funções e chamadas de métodos
- VAR\_TEMP/VAR\_STAT/VAR\_RETAIN/ VAR\_PERSISTENT
- Expressões arbitrárias para inicialização de variáveis
- Atribuição como expressão
- Acesso de índice com ponteiros e strings

### Conceito de Gerenciamento de Usuário e Direitos de Acesso

- Contas de usuários, grupos de usuários, direitos específicos de grupos para acesso e ações em objetos específicos

### Características em Editores

- Editor ST: recursos de edição, quebra de linha, autocompletar, monitoração e atribuição SET/RESET na linha
- Editores FBD, LD e IL reversíveis e programáveis em um editor combinado
- Editor IL como editor de tabela
- Editores FBD, LD e IL: possibilidade de alteração da saída principal em caixas com várias saídas
- Editores FBD, LD e IL sem atualização automática dos parâmetros da caixa
- Editores FBD, LD e IL: ramificações e redes dentro de redes
- Editor SFC: somente um tipo de passo, macros, seleção múltipla de elementos independentes, sem verificação sintática durante a edição e declaração automática de variáveis sinalizadoras

### Versões de Bibliotecas

- Várias versões de bibliotecas podem ser usadas no mesmo projeto utilizando o recurso de contextos
- Instalação em repositórios, atualização automática e depuração

### Funcionalidades Adicionais

- Menus, barra de ferramentas e uso de teclado
- Possibilidade de incluir componentes específicos do usuário
- Configuração do CP e configuração de tarefas integradas na árvore de dispositivos
- Suporte a UNICODE
- Comentários de linha
- Cão-de-Guarda
- Seleção múltipla na árvore de objetos do projeto
- Ajuda online integrada na interface do usuário
- Compilação condicional
- Breakpoints condicionais

- Depuração: passo para o cursor e retorno à chamada anterior
- Driver de barramento de campo em conformidade com a norma IEC 61131-3
- Configuração do CP e de símbolos disponíveis na aplicação
- Alocação livre de memória de código e dados
- Cada objeto pode ser especificado como “interno” ou “externo” (link posterior no sistema de execução)
- Notas de pré-compilação referentes a erros sintáticos

### Perfis

Um perfil de projeto no MasterTool IEC XE é um conjunto de regras, características comuns e padrões utilizados no desenvolvimento de uma solução de automação industrial, um perfil que influencia a forma de implementação da aplicação. Com a diversidade de tipos de aplicações suportadas pelo Runtime System da Série Nexto, seguir um perfil é uma forma de reduzir a complexidade na programação.

As aplicações podem ser criadas conforme um dos seguintes perfis:

- Simples
- Básico
- Normal
- Experiente
- Personalizado

O software MasterTool IEC XE disponibiliza um template, denominado *Projeto MasterTool Padrão*, o qual deve ser selecionado pelo usuário como modelo na criação de um projeto. A nova aplicação será desenvolvida conforme um determinado perfil, também escolhido pelo usuário, adotando as regras, características e padrões pré-definidos. Cada perfil de projeto define nomes padronizados para tarefas e programas.

Para garantir a compatibilidade de um projeto a um determinado perfil ao longo do desenvolvimento, são utilizadas duas abordagens:

- O MasterTool IEC XE somente permite a criação de projetos baseados em um template, selecionando ao mesmo tempo o perfil a ser utilizado.
- Na geração de código, o MasterTool IEC XE realiza a verificação de todas as regras definidas para o perfil válido para o projeto.

Para maiores detalhes sobre os perfis consultar seção **Perfis** no Manual de Utilização UCPs Série Nexto – MU214100, no capítulo Programação Inicial.

### Projeto

Um projeto contém os objetos das POU's que compõem um programa do CP, assim como as definições dos objetos de recursos necessários para executar uma ou mais instâncias do programa (aplicação) em determinados sistemas-destino (CPs, dispositivos). Objetos de POU's podem ser gerenciados na janela de visualização das POU's ou na janela de visualização dos dispositivos, POU's criadas a partir do wizard, aparecem na janela visualização dos dispositivos, e os objetos de recursos específicos do dispositivo são gerenciados na janela de visualização dos dispositivos.

Um projeto é salvo no arquivo <nome do projeto>.project.

NOTA: A aparência e as propriedades da interface do usuário são definidas e armazenadas no MasterTool IEC XE e não no projeto.
--

### Dispositivos

Na janela *Dispositivos* (“árvore de dispositivos”) define-se o hardware onde a aplicação será executada.


Cada “dispositivo” representa um hardware específico (destino). Exemplos: controlador, módulos de E/S, monitor.

Cada dispositivo é definido por uma descrição e deve ser instalado no sistema local para que possa ser inserido na árvore de *Dispositivos*. O arquivo de descrição define as propriedades referentes à configuração, programação e possíveis conexões com outros dispositivos.

Na árvore de *Dispositivos* são gerenciados os objetos necessários para executar uma aplicação no dispositivo (controlador, CP), incluindo aplicação, configuração de tarefas e tarefas. Entretanto, objetos de programação específicos (POUs, listas de variáveis globais e gerenciador de bibliotecas) podem - em vez de serem gerenciados como unidades instanciáveis globais de projeto na janela das POU's - serem gerenciados SOMENTE na árvore de *Dispositivos* e, neste caso, estarem disponíveis apenas na sua aplicação ou nas suas “aplicações secundárias”.

### Aplicação

Uma “aplicação” é o conjunto dos objetos necessários para executar uma instância específica do programa do CP em um determinado dispositivo de hardware (CP, controlador). Para isto, objetos “independentes” gerenciados na visualização das POU's são instanciados e atribuídos a um dispositivo na janela visualização dos *Dispositivos*. Isto está em conformidade com a programação orientada a objetos. Entretanto, POU's específicas da aplicação também podem ser utilizadas.


Uma aplicação é representada por um objeto de aplicação () na árvore de *Dispositivos* inserido abaixo de um nó do dispositivo programável (PLC Logic). Os objetos que definem o “conjunto de recursos” da aplicação podem ser inseridos abaixo de um item da aplicação.

A aplicação padrão, “Application”, é criada junto com novos projetos criados a partir do modelo *Projeto MasterTool Padrão* ela é acrescentada à árvore de dispositivos abaixo do item *Device* e *PLC Logic*.

Uma parte essencial da aplicação é a Task Configuration que controla a execução de um programa (instâncias de POU ou POU's específicas da aplicação). Adicionalmente, podem estar atribuídos objetos de recursos, tais como listas de variáveis globais, bibliotecas, entre outros, os quais - ao contrário daqueles gerenciados na janela das POU's - somente podem ser usados pela aplicação específica e seus subitens.

A compatibilidade dos parâmetros da aplicação no CP com os parâmetros da aplicação do projeto é verificada no momento do login em um dispositivo (CP ou dispositivo de simulação). Em caso de incompatibilidade, é exibida uma mensagem apropriada.

### Task Configuration

A Task Configuration () define uma ou várias tarefas para controlar o processamento de um programa aplicativo.

Ela é um objeto de recurso essencial para uma aplicação e é inserido automaticamente ao criar um novo projeto a partir do modelo *Projeto MasterTool Padrão*. Uma tarefa pode chamar uma POU de programa específica da aplicação que esteja disponível na árvore de dispositivos, assim como o programa gerenciado na janela das POU's. Neste último caso, o programa do projeto global disponível será instanciado pela aplicação.

Uma task configuration pode ser editada no editor de tarefas, sendo as opções disponíveis específicas do dispositivo.

No modo online o editor de tarefas fornece uma visualização da monitoração e também informações sobre os ciclos, tempos e status.

### Notas Importantes para Sistemas Multitarefa

Em alguns sistemas são realizadas algumas multitarefas preferenciais. Neste caso, as seguintes observações devem ser consideradas.

Todas as tarefas compartilham o mesmo mapa de processo, pois um mapa para cada tarefa prejudicaria a performance. Entretanto, o mapa do processo sempre consiste de apenas uma tarefa. Assim, ao criar um projeto, o usuário deve explicitamente observar que, em caso de conflitos, os dados de entrada serão copiados para a área de salvamento (o mesmo ocorre com as saídas). Por exemplo, módulos da biblioteca “SysSem” podem ser usados para resolver problemas de sincronização.

Ao acessar outros objetos globais (variáveis globais, módulos), podem ocorrer problemas de consistência, se o tamanho dos objetos excederem a capacidade do processador (estruturas ou ARRAYS formando uma unidade lógica). Neste caso, os módulos da biblioteca “SysSem” também podem ser usados para resolver os problemas.

### Comunicação

Para informações sobre **Comunicação** (Configuração do CP, Topologia de rede, Endereçamento e roteamento, Estrutura de endereços e Variáveis de rede), consulte o Manual de Utilização MasterTool IEC XE – MU299048.

### Geração de Código e Alteração Online

#### Geração de Código e Informações de Compilação

O código fonte não será gerado até que o projeto da aplicação seja enviado para o dispositivo (CP ou dispositivo de simulação). A cada envio, as informações de compilação contendo o código e a ID de referência da aplicação carregada serão armazenadas em um diretório do projeto em um arquivo “<nome do projeto>.<nome do dispositivo>.<ID da aplicação>.compileinfo”. As informações de compilação serão apagadas quando os comandos *Limpar* e *Limpar Tudo* forem executados.

#### Alterações Online

Dependendo da alteração no projeto em execução no controlador, apenas os objetos modificados serão carregados para o mesmo.

##### ATENÇÃO:

Alterações online modificam o programa aplicativo em execução e não provocam a reinicialização do mesmo. Certifique-se de que o novo código da aplicação não afetará o comportamento esperado do sistema. Dependendo do controlador, podem ocorrer danos nas máquinas ou nas suas partes, assim como pode haver risco à saúde e à vida das pessoas.

##### NOTAS:

- Quando é realizada uma alteração online, as inicializações específicas da aplicação não serão executadas, pois a máquina mantém o seu estado. Por esta razão, o novo código de programa pode não funcionar conforme o desejado.
- Variáveis de ponteiro mantêm os valores do ciclo anterior. Se houver um ponteiro em uma variável que tenha alterado seu tamanho devido a uma alteração online, o valor não estará mais correto. Certifique-se de que as variáveis de ponteiro sejam re-atribuídas a cada ciclo.

#### Aplicação de Inicialização (Projeto de Inicialização)

Uma aplicação de inicialização é o projeto que será iniciado automaticamente quando o controlador é inicializado. Para tanto, o projeto deve estar disponível no CP em um arquivo “<nome do projeto>.app”. Este arquivo pode ser criado no modo offline através do comando *Criar Aplicação de Inicialização* (menu *Comunicação*).

A cada envio bem sucedido, a aplicação ativa será automaticamente armazenada no arquivo “<aplicação>.app” na pasta do sistema do dispositivo, ficando assim disponível como uma aplicação

de inicialização. O comando *Criar Aplicação de Inicialização* também permite salvar esta aplicação em um arquivo no modo offline.

### Método de Envio/Login de Projetos Sem Diferença de Projetos

Afim de garantir que o usuário não terá problemas ao enviar projetos iguais e logar em UCPs em execução a partir de diferentes estações, podem ser executados os seguintes passos após o envio de um projeto:

- No diálogo *Arquivos Adicionais* (menu *Projeto, Configuração do Projeto, Download de Código Fonte* e botão *Arquivos Adicionais*) marcar a opção *Realizar download dos arquivos de informações*.
- Fechar todos diálogos clicando em *OK*.
- Executar o comando *Download do Código Fonte* (menu *Arquivo*).
- No diálogo *Selecionar Dispositivo*, que será aberto, escolher a UCP em que o projeto foi enviado.
- Fechar o diálogo clicando em *OK*, aguardar o download do projeto.

Para logar em UCPs em execução sem gerar alterações de projeto a partir de diferentes estações, deve-se abrir um arquivo de projeto gerado a partir do projeto original e executar o comando *Login*. Na falta deste, podem ser realizados os seguintes procedimentos:

- Executar o comando *Carregar Código Fonte* (menu *Arquivo*).
- No diálogo *Selecionar Dispositivo*, que será aberto, escolher a UCP em que o projeto foi enviado.
- Fechar o diálogo clicando em *OK*, aguarde o carregamento do projeto.
- No diálogo *Arquivo de Projeto*, que será aberto ao fim do processo de carregamento, escolher o local para extração e clicar no botão *Extrair*.
- O projeto será aberto e o comando de *Login* pode ser executado na UCP correspondente.

Para mais informações consulte: **Menu Arquivo** e **Menu Comunicação** no Manual de Utilização MasterTool IEC XE – MU299048.

### Monitoração

No modo online, existem várias possibilidades de exibir os valores atuais das expressões de monitoração de um objeto no CP.

Para mais informações a respeito deste item, consultar **Monitoração** no Manual de Utilização MasterTool IEC XE – MU299048.

### Depuração

Para avaliar erros de programação use a funcionalidade de depuração do MasterTool IEC XE no modo online. Neste contexto, considere a possibilidade de verificar a aplicação no modo de simulação, isto é, sem a necessidade de conectar-se com um dispositivo de hardware real.

Breakpoints podem ser configurados em determinadas posições para forçar uma interrupção na execução. Algumas condições podem ser configuradas para cada breakpoint, como por exemplo, quais as tarefas associadas ou em quais ciclos o breakpoint deve atuar. Funções de passos estão disponíveis para que um programa seja executado em passos controlados. A cada interrupção, os valores atuais das variáveis podem ser examinados. Uma pilha de chamadas pode ser visualizada na posição do passo atual.

Para mais informações a respeito deste item, consultar **Breakpoints** no Manual de Utilização MasterTool IEC XE – MU299048.

### Linguagens de Programação Suportadas

Todas as linguagens de programação mencionadas na norma IEC 61131 são suportadas através de editores especialmente adaptados.



- Editores FBD/LD/IL para Diagrama de Bloco Funcional (FBD), Diagrama Ladder (LD) e Lista de Instruções (IL)
- Editor SFC para Sequenciamento Gráfico de Funções
- Editor ST para Texto Estruturado

O MasterTool IEC XE fornece adicionalmente um editor para programação em CFC, denominado Editor CFC para Gráfico Funcional Contínuo. Este, entretanto, não faz parte da norma IEC.

### Unidades de Organização de Programas

O termo POU é utilizado basicamente para todos os objetos usados para criar um programa do CP, e significa Unidade de Organização de Programa.

POUs gerenciadas na Visualização das POU's não são específicas do dispositivo, mas devem ser instanciadas para uso em um dispositivo (aplicação). Para tanto, POU's de programa devem ser chamadas por uma tarefa da respectiva aplicação.

As POU's inseridas na árvore de *Dispositivos* explicitamente em uma aplicação SOMENTE são gerenciadas na Visualização dos Dispositivos, ou seja, podem ser instanciadas apenas por aplicações recuadas abaixo desta aplicação (aplicação secundária). Para obter mais informações, veja as informações sobre **Árvore de Dispositivos** e **Aplicação** no Manual de Utilização MasterTool IEC XE – MU299048.

No entanto, POU também é o nome de uma determinada subcategoria destes objetos no menu *Acrescentar Objeto*, que aí compreende programas, blocos funcionais e funções.

Assim sendo, uma POU - Unidade de Organização de Programa, em geral, é uma unidade de programação, um objeto gerenciado através de dispositivos não específicos (na janela das *POU's*) e específicos (na janela dos *Dispositivos*) e que pode ser visualizado e editado em uma janela do editor. Uma POU pode ser uma função, bloco funcional, método, ação, DUT ou ainda um arquivo externo de qualquer formato.

Considere a possibilidade de configurar determinadas propriedades (como por exemplo, condições de compilação, etc.) da POU.


Por padrão, os seguintes tipos de POU podem ser usados:

- POU
- Ação
- DUT (Unidade de Tipo de Dado)
- Arquivo Externo
- Lista de Variáveis Globais
- Método
- Propriedades
- Programa
- Função
- Bloco Funcional
- Variáveis Persistentes
- POU's para Verificações Implícitas

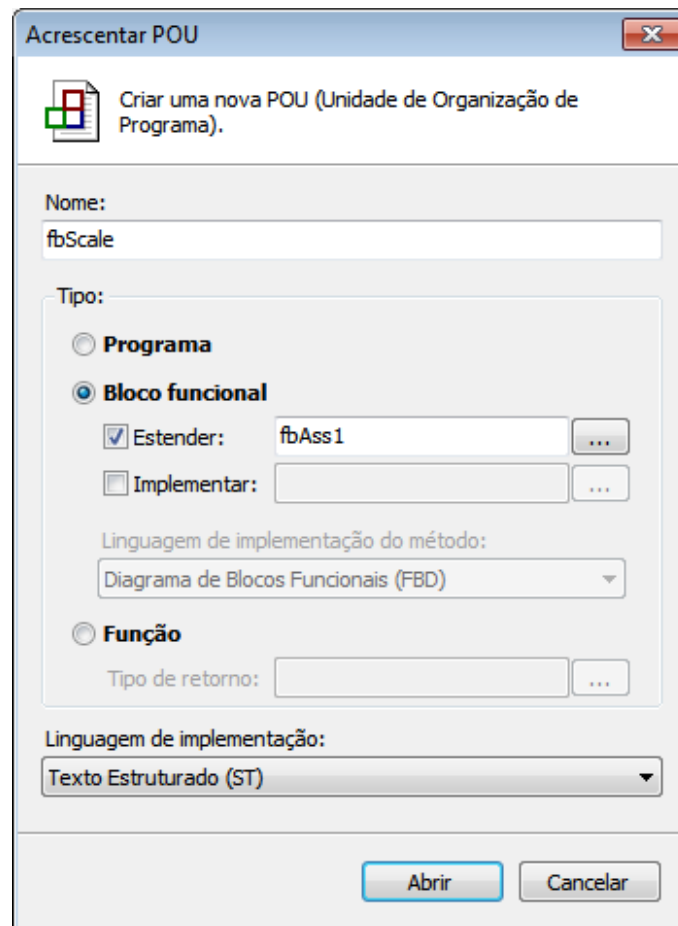
Além da Unidade de Organização de Programa, existem os Dispositivos, os quais são usados para executar o programa (recurso, aplicação, configuração de tarefa, etc.). Estes últimos são gerenciados na visualização dos *Dispositivos*.

### POU

Uma POU é uma Unidade de Organização de Programa do tipo Programa, Função ou Bloco Funcional.

Para acrescentar uma POU () , selecione o item correspondente na janela *POU's* ou *Dispositivos* (uma aplicação, por exemplo), use o comando *Acrescentar Objeto* no menu de contexto e selecione

POU no submenu que aparecerá. O diálogo *Acrescentar POU* abrirá para que se configure o seu nome, tipo e linguagem de implementação. No caso dos blocos funcionais, opcionalmente podem ser definidas as propriedades **ESTENDER** e **IMPLEMENTAR**. No caso das funções, o tipo de retorno também deve ser especificado. Veja os respectivos itens em **Programa**, **Função** e **Bloco Funcional**.



**Figura 2-1. Diálogo Acrescentar POU**

Dependendo do tipo, a POU pode ser complementada por métodos, propriedades, ações e transições. Para tal, também é utilizado o comando *Acrescentar Objeto*.

A ordem hierárquica de processamento das instâncias da POU de uma aplicação depende da configuração específica do dispositivo (pilha de chamadas).

Cada POU consiste de uma parte de declaração e uma parte de implementação. O corpo é escrito em uma das linguagens de programação disponíveis, que são IL, ST, SFC, FBD, LD ou CFC.

O MasterTool IEC XE suporta todas as POU's descritas pela norma IEC 61131-3. Para usar estas POU's no projeto, deve-se incluir a biblioteca standard.library. Os projetos criados a partir do modelo *Projeto MasterTool Padrão* já possuem esta biblioteca carregada.

NOTA: Em alguns exemplos deste manual o código está declarado sequencialmente, mas para sua utilização o mesmo deve ser separado, a parte superior do editor da linguagem de programação para declarações e a parte inferior do editor deve ser utilizada para a implementações.

### Chamando POU's

POU's podem chamar outras POU's, mas não são permitidas recorrências.

Quando uma POU atribuída a uma aplicação chama outra POU apenas pelo seu nome (sem nenhum contexto adicionado), é adotada a seguinte ordem de busca:

1. Aplicação atual
2. Gerenciador de bibliotecas da aplicação atual
3. Visualização das POU's
4. Gerenciador de bibliotecas na visualização das POU's

Se uma POU com o nome especificado na chamada estiver disponível em uma biblioteca do gerenciador de bibliotecas e estiver como um objeto na visualização das POU's, não haverá sintaxe para chamá-las explicitamente apenas usando o seu nome na visualização das POU's. Neste caso, o usuário deve mover a referida biblioteca do gerenciador da aplicação para o gerenciador da visualização das POU's. Isso permitirá a chamada da POU apenas pelo seu nome. Por outro lado, para chamar aquela da biblioteca, deve-se utilizar o contexto da referida biblioteca.

Consulte também o item **POU's para Verificações Implícitas**.

### Programa

Um programa é uma POU que retorna um ou vários valores durante a operação. Todos os valores são mantidos desde a última vez que o programa foi executado até a próxima execução.

Uma POU de programa pode ser adicionada ao projeto via comando *Acrescentar Objeto*. Para atribuir o programa a uma aplicação existente, selecione-a na visualização dos *Dispositivos* e use o comando a partir do menu de contexto, caso contrário, a POU será adicionada à visualização das *POU's*. No diálogo *Acrescentar POU*, escolha *Programa*, digite um nome para ele e defina a linguagem desejada. Após confirmar as configurações via botão *Abrir*, a janela do editor para o novo programa será aberta para que se possa iniciar a sua edição.

Sintaxe para declarar um programa:

```
PROGRAM <NOME DO PROGRAMA>
```

Abaixo do nome, seguem as declarações de variáveis de entrada, saída e variáveis de programa. Opcionalmente, também podem ser declaradas variáveis de acesso.

```
1 PROGRAM PRGexample
2 VAR_INPUT
3   in_var:INT;
4 END_VAR
5 VAR_OUTPUT
6   out_var:INT;
7 END_VAR
8 VAR
9   ivar:INT;
10  bvar:BOOL;
11 END_VAR
12 out_var:=in_var+ivar;
13 IF out_var:=23;
14   THEN bvar:=TRUE;
15 END_IF;
```

Figura 2-2. Exemplo de Programa

### Chamada de Programas

Um programa pode ser chamado por outra POU, mas a chamada de programa não é permitida em uma função. Também não existem instâncias de programas.

Se uma POU chamou um programa e se os valores deste foram alterados, estas alterações serão mantidas até que o programa seja chamado novamente (mesmo que seja chamado em outra POU). Note que isto é diferente de chamar um bloco funcional, onde somente os valores na instância fornecida são alterados. Assim, estas alterações só têm efeito quando a mesma instância for chamada novamente.

Os parâmetros de entrada e/ou saída no curso de uma chamada de programa podem ser definidos nos editores de linguagem textual (ST, por exemplo), atribuindo-se valores a eles após o nome do programa entre parênteses. Para parâmetros de entrada, esta atribuição ocorre usando "=", assim como com a inicialização das variáveis na posição de declaração. Para parâmetros de saída, deve ser usado "=>". Veja o exemplo abaixo.

Se o programa for inserido via *Assistente de Entrada* com a opção *Inserir com argumentos* na janela de implementação de um editor de linguagem textual, ele será exibido automaticamente de acordo com esta sintaxe para todos os parâmetros (embora estes não necessariamente tenham que ser atribuídos).

Exemplo para chamada de programa em IL:

```

CAL          PRGexample      (
           in_var:= 33      )
LD          PRGexample.out_var
ST          erg
    
```

Atribuindo os parâmetros (utilizando *Assistente de Entrada* com opção *Com argumentos* marcada):

```

CAL          PRGexample      (
           in_var:= 33      ,
           out_var=> erg    )
    
```

Exemplo em ST:

```

PRGEXAMPLE ();
ERG := PRGEXAMPLE.OUT_VAR;
    
```

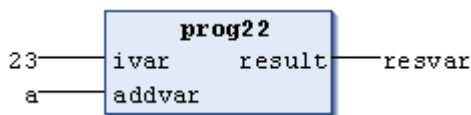
Atribuindo os parâmetros (*Assistente de Entrada* com opção *Com argumentos*):

```

PRGEXAMPLE (IN_VAR:=33, OUT_VAR=>ERG);
    
```

NOTA: Os parênteses são obrigatórios.

Exemplo em FBD:



### Função

Uma função é uma POU que produz exatamente um elemento de dados (que pode consistir de um ou vários elementos, tais como campos ou estruturas) ao ser processada e cuja chamada em linguagens textuais pode ocorrer como um operador em expressões.

Via comando *Acrescentar Objeto* e *Acrescentar POU*, pode-se adicionar uma função ao projeto. Para atribuí-la a uma aplicação existente, selecione-a na visualização dos *Dispositivos* e use o comando do menu de contexto. Caso contrário, ela será adicionada à visualização das *POUs*. No diálogo *Acrescentar POU*, escolha o tipo *Função*, digite um nome (<nome da função>) e o tipo de dados de

retorno (<tipo de dados>) para a nova função e escolha a linguagem de implementação desejada. O editor estará disponível via botão *Abrir*, permitindo, na sequência, a edição da POU.

Declaração:

A declaração da função inicia com a palavra-chave FUNCTION. Devem ser definidos um nome e um tipo de dados.

Sintaxe:

```
FUNCTION <NOME DA FUNÇÃO> : <TIPO DE DADOS>
```

Esta sintaxe é seguida pelas declarações das variáveis de entrada e das variáveis da função.

Um resultado deve ser atribuído a uma função, ou seja, o nome da função é usado como variável de saída.

NOTA: Se uma variável local for declarada como retentiva em uma função, isto não terá efeito. A variável não será incluída na área de retenção.

Exemplo de uma Função em ST:

```
FUNCTION FCT : INT
VAR_INPUT
IVAR1 : INT;
IVAR2 : INT;
IVAR3 : INT;
END_VAR
FCT:=IVAR1+IVAR2*IVAR3;
```

Esta função lê três variáveis de entrada e retorna o produto das duas últimas, adicionadas à primeira.

### Chamada de Função

A chamada de uma função em ST pode aparecer como um operando em expressões.

Em IL, uma chamada de função somente pode estar posicionada dentro das ações de um passo ou dentro de uma transição.

Funções (ao contrário dos programas ou blocos funcionais) não contêm informações de estado, ou seja, a invocação de uma função com os mesmos argumentos (parâmetros de entrada) sempre produzirá os mesmos valores (saída). Por esta razão, as funções não devem conter variáveis globais e endereços.

Abaixo, encontram-se exemplos para chamadas de função.

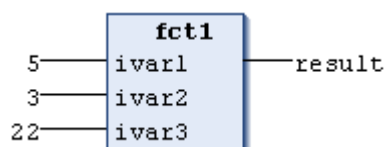
Em IL:

<b>LD</b>	5	
<b>Fct</b>	3	r
	22	
<b>ST</b>	result	

Em ST:

```
RESULT := FCT1(5, 3, 22);
```

Em FBD:



Em chamadas de função, não é possível misturar atribuições de parâmetros explícitos com implícitos. Isto permite alterar a ordem das atribuições dos parâmetros de entrada.

Exemplo:

```
FUN(FORMAL1 := ACTUAL1, ACTUAL2); // -> Mensagem de erro.  
FUN(FORMAL2 := ACTUAL2, FORMAL1 := ACTUAL1); // Mesma semântica que:  
FUN(FORMAL1 := ACTUAL1, FORMAL2 := ACTUAL2);
```

De acordo com a norma IEC 61131-3, as funções podem ter saídas adicionais. Estas saídas devem ser atribuídas na chamada de uma função, por exemplo, em ST, conforme a seguinte sintaxe:

```
OUT1 => <VARIÁVEL DE SAÍDA 1> | OUT2 => <VARIÁVEL DE SAÍDA 2> |... OUTRAS  
VARIÁVEIS DE SAÍDA
```

Exemplo:

A função FUN é definida com duas variáveis de entrada in1 e in2. O valor de retorno de FUN será escrito nas variáveis de saída (VAR\_OUTPUT) loc1 e loc2 declaradas localmente.

```
FUN(IN1 := 1, IN2 := 2, OUT1 => LOC1, OUT2 => LOC2);
```

### Bloco Funcional

Um bloco funcional é uma POU que fornece um ou mais valores durante o processamento de um programa do CP. Ao contrário da função, os valores das variáveis de saída e das variáveis internas necessárias mantêm-se de uma execução do bloco funcional até a próxima execução. Assim sendo, a chamada de um bloco funcional com os mesmos argumentos (parâmetros de entrada) nem sempre produz os mesmos valores de saída.

Além da funcionalidade descrita na norma IEC 61131-3, a programação orientada a objetos é suportada e os blocos funcionais podem ser definidos como extensões de outros blocos funcionais. Isto significa que a herança pode ser usada ao programar com blocos funcionais.

Um bloco funcional sempre é chamado através de uma instância, que é uma reprodução (cópia) do bloco funcional.

Via comando *Acrescentar Objeto* e *Acrescentar POU*, um bloco funcional pode ser adicionado ao projeto. Para atribuí-lo a uma aplicação existente, selecione-o na visualização dos dispositivos e use o comando do menu de contexto. Caso contrário, ele será adicionado à visualização das POUs.

No diálogo *Acrescentar POU*, escolha o tipo *Bloco Funcional*, digite o nome do bloco (<identificador>) e escolha a linguagem de implementação.

Opções adicionais:

- *Estender*: digite o nome de outro bloco funcional disponível no projeto, que deve ser a base do bloco atual (corrente).
- *Implementar*: não é suportado.

O editor estará disponível via botão *Abrir*, permitindo, na sequência, a edição da POU.

Declaração:

Sintaxe:

```
FUNCTION_BLOCK <NOME DO BLOCO FUNCIONAL> | EXTENDS <NOME DO BLOCO  
FUNCIONAL>
```

A sintaxe acima é seguida pela declaração das variáveis.

NOTA: FUNCTION BLOCK não é uma palavra-chave alternativa válida.
--

Exemplo:

O FBexample, mostrado na figura a seguir, apresenta duas variáveis de entrada e duas de saída, out1 e out2. Out1 é a soma das duas entradas, out2 é o resultado de uma comparação de igualdade.

Exemplo de um Bloco Funcional em ST:

```
FUNCTION_BLOCK FBEXAMPLE
VAR_INPUT
INP1: INT;
INP2: INT;
END_VAR
VAR_OUTPUT
OUT1: INT;
OUT2: BOOL;
END_VAR
OUT1 := INP1 + INP2;
OUT2 := INP1 = INP2;
```

### *Instância de Bloco Funcional*

Blocos funcionais são sempre chamados através de uma instância, a qual é uma reprodução (cópia) de um bloco funcional.

Cada instância tem o seu próprio identificador (nome da instância) e uma estrutura de dados contendo entradas, saídas e variáveis internas.

Assim como as variáveis, as instâncias são declaradas local ou globalmente e, por meio delas, o nome do bloco funcional é indicado como o tipo de dado de um identificador.

Sintaxe para declaração de uma instância de bloco funcional:

```
<identificador>: <nome do bloco FUNCIONAL>;
```

Exemplo:

Declaração (por exemplo, na parte de declaração de um programa) da instância INSTANCE do bloco funcional FUB:

```
INSTANCE: FUB;
```

As partes de declaração dos blocos funcionais e programas podem conter declarações de instâncias, no entanto, nas funções, as declarações de instâncias não são permitidas.

### *Chamando um Bloco Funcional*

Instâncias de blocos funcionais devem ser declaradas local ou globalmente.

A variável do bloco funcional pode ser acessada usando-se a sintaxe abaixo.

Sintaxe:

```
<NOME DA INSTÂNCIA>.<NOME DA VARIÁVEL>
```

Considere:

- Somente as variáveis de entrada e saída de um bloco funcional podem ser acessadas externamente a uma instância do bloco funcional. Suas variáveis internas não são acessíveis.
- O acesso a uma instância do bloco funcional é limitado à POU na qual ela foi declarada, a menos que tenha sido declarada globalmente.
- Na chamada da instância, os valores desejados podem ser atribuídos aos parâmetros dos blocos funcionais.
- As variáveis de entrada e saída (VAR\_IN\_OUT) de um bloco funcional são passadas como ponteiros.
- No SFC, as chamadas de blocos funcionais somente podem ocorrer em ações associadas aos passos.
- O nome da instância do bloco funcional pode ser usado como um parâmetro de entrada para uma função ou outro bloco funcional.
- Todos os valores de um bloco funcional ficam retidos até o próximo processamento do mesmo, portanto, as chamadas de blocos funcionais nem sempre retornam os mesmos valores de saída, mesmo que tenham sido utilizados os mesmos argumentos.

NOTA: Se no mínimo uma das variáveis do bloco funcional for retentiva, a instância total é armazenada na área de dados retentivos.

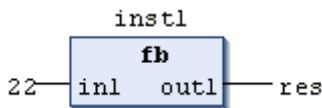
Abaixo, encontram-se exemplos para acessar variáveis de blocos funcionais.

Suponha que o bloco funcional fb tem uma variável de entrada in1 do tipo INT. Veja a chamada desta variável a partir do programa prog.

Declaração e implementação em ST:

```
PROGRAM PROG
VAR
INST1:FB;
RES:INT;
END_VAR
INST1.IN1:=22; (* fb é chamado e a variável in1 assume o valor 22 *)
INST1(); (* fb é chamado, isto é necessário para o acesso à variável de
saída mostrado a seguir *)
RES:=FBINST.OUTL; (* Variável de saída do fb é lida *)
```

Exemplo em FBD:



### Atribuindo Parâmetros na Chamada

Nas linguagens textuais IL e ST, os parâmetros de entrada/saída podem ser configurados imediatamente na chamada de um bloco funcional. Os valores podem ser atribuídos aos parâmetros entre parênteses após o nome da instância do bloco funcional. Para parâmetros de entrada, esta atribuição ocorre usando “:=”, assim como ocorre com a inicialização das variáveis na declaração. Para parâmetros de saída, deve ser usado “=>”.

Veja um exemplo de chamada com atribuições, onde um bloco funcional de temporização (instância CMD\_TMR) é chamado com atribuições para os parâmetros IN e PT. A variável resultante Q é atribuída à variável A com o nome da instância do bloco funcional, seguido de um ponto e do nome da variável:

```
CMD_TMR(IN := %IX5.0, PT := 300);
A:=CMD_TMR.Q
```

Se a instância é inserida via *Assistente de Entrada* (<F2>), através da opção *Inserir com Argumentos* na janela de implementação de uma POU ST ou IL, ela será automaticamente exibida de acordo com a sintaxe mostrada a seguir, com todos os seus parâmetros, mas não necessariamente deve atribuir estes parâmetros. Para o exemplo acima, a chamada deve aparecer da seguinte forma:

Exemplo de inserção via *Assistente de Entrada* com argumentos:

```
CMD_TMR(in:=, pt:=, q=>) (* Nenhum parâmetro atribuído *)
```

->

```
CMD_TMR(in:=bvar, pt:=t#200ms, q=>bres); (* Parâmetros atribuídos *)
```

### Extensão de um Bloco Funcional

A programação orientada ao objeto permite a derivação de um bloco funcional a partir de outro. Desta forma, as propriedades de um bloco funcional podem ser estendidas a outro.

A extensão é feita usando-se a palavra-chave EXTENDS na declaração do bloco funcional. É possível escolher a opção EXTENDS ao acrescentar o bloco ao projeto através do diálogo *Acréscimo de Objeto*.

Sintaxe:



```
FUNCTION_BLOCK <nome do bloco funcional> EXTENDS <nome do bloco funcional>
```

A declaração das variáveis é então realizada na sequência.

Exemplo de definição do bloco funcional fbA:

```
FUNCTION_BLOCK fbA
VAR_INPUT
x:int;
...
```

Definição do bloco funcional fbB:

```
FUNCTION_BLOCK fbB EXTENDS fbA
VAR_INPUT
ivar:int;
...
```

No exemplo acima:

- FbB contém todos os dados e métodos definidos por fbA. Uma instância de fbB pode agora ser usada em qualquer contexto onde um bloco funcional do tipo fbA é esperado.
- FbB pode substituir os métodos definidos em fbA. Isto significa que fbB pode declarar um método com o mesmo nome e as mesmas entradas e saídas declaradas em fbA.
- Não é permitido que fbB use as variáveis do bloco funcional com o mesmo nome usado em fbA. Neste caso, o compilador apontará um erro.
- Variáveis e métodos de fbA podem ser acessados diretamente dentro do escopo fbB através da palavra-chave SUPER (SUPER^.<método>).

NOTA: Não são permitidas heranças múltiplas. Ou seja, não é possível estender mais do que um bloco de função.

### Invocação de Método

A programação orientada ao objeto, quando aplicada em blocos funcionais, além da opção ESTENDER, permite o uso de herança. Ela requer invocações dinâmicas de método, conhecidas como chamadas de função virtual.

Para obter maiores informações, consulte o item **Método**.

Chamadas de função virtual precisam de mais tempo que as chamadas de função normais e são usadas quando:

- Uma chamada é realizada através de um ponteiro para um bloco funcional (pfub^.method)
- Um método chama outro método do mesmo bloco funcional

Chamadas de função virtual possibilitam que a mesma chamada em um código fonte do programa invoque diferentes métodos durante a execução.

De acordo com a norma IEC 61131-3, métodos como funções normais podem ter saídas adicionais. Estas devem ser atribuídas na chamada do método, conforme a seguinte sintaxe:

```
<MÉTODO>(IN1:=<VALOR> |, ATRIBUIÇÕES DE ENTRADA ADICIONAIS, OUT1 =>
<VARIÁVEL DE SAÍDA 1> | OUT2 => <VARIÁVEL DE SAÍDA 2> | ...VARIÁVEIS DE
SAÍDA ADICIONAIS)
```

Isto faz com que a saída do método seja escrita nas variáveis de saída declaradas localmente, conforme definido na chamada.

Exemplo:

Suponha que os blocos funcionais fub1 e fub2 ESTENDEM o bloco funcional fubbase. O método method1 está incluído.

Possível uso das chamadas de método:

```
VAR_INPUT
```

```
B : BOOL;
END_VAR
VAR
  PINST : POINTER TO FUBBASE;
  INSTBASE : FUBBASE;
  INST1 : FUB1;
  INST2 : FUB2;
END_VAR
IF B THEN
  PINST := ADR(INSTBASE);
ELSE
  PINST := ADR(INST1);
END_IF
PINST^.METHOD1(); (* Se B é TRUE, FUBBASE.METHOD1 é chamado, senão
FUB1.METHOD1 é chamado *)
```

Agora, suponha que fubbase do exemplo acima contém dois métodos, method1 e method2. fub1 inclui method2 mas não method1. method1 é chamado, conforme mostrado no exemplo acima, como segue:

```
PINST^.METHOD1(); (*Se B é TRUE, FUBBASE.METHOD1 é chamado, senão
FUB1.METHOD1 é chamado*)
```

Chamada via ponteiros - Implementação de method1:


```
METHOD METHOD1 : BOOL
VAR_INPUT
END_VAR
METHOD1 := METHOD2();
```

Chamando fubbase.method1 via ponteiros, ocorre:

- Se o ponteiro for do tipo fubbase, fubbase.method2 será chamado
- Se o ponteiro for do tipo fub1, fub1.method2 será chamado

### Unidades de Tipo de Dados

Além dos tipos de dados padrão, o usuário pode utilizar tipos criados por ele. Tipos de estruturas, enumeração e referências podem ser criados como Unidades de Tipos de Dados (DUT) em um editor DUT.

Uma DUT () pode ser adicionada ao projeto através do comando *Acréscimo Objeto*. Para atribuí-la a uma aplicação existente, selecione a aplicação na árvore de *Dispositivos*. Caso contrário, a DUT será adicionada à árvore das *POUs*. No diálogo *Acréscimo DUT*, digite um nome para ela (nova unidade de tipos de dados) e escolha o tipo desejado de *Estrutura*, *Enumeração*, *Pseudônimo* ou *União*.

No caso do tipo *Estrutura*, é possível utilizar o princípio da herança como suporte à programação orientada ao objeto. Opcionalmente, pode ser especificado que uma DUT estenda-se à outra DUT que esteja definida no projeto. Isto significa que as definições da DUT estendida serão automaticamente válidas dentro da DUT atual. Para este propósito, ativa-se a opção *Estender* e digita-se o nome da outra DUT.

Após confirmar a configuração, pressione o botão *Abrir*. O editor da nova DUT estará disponível e pode-se iniciar a edição.

Sintaxe para declaração de uma DUT:

```
TYPE <IDENTIFICADOR> : <DECLARAÇÃO DOS COMPONENTES DA DUT>
END_TYPE
```

A declaração dos componentes da DUT depende do seu tipo, por exemplo, se é uma estrutura ou uma enumeração.

Exemplo:

Veja nas duas DUTs abaixo, estruturas de definição struct1 e struct, sendo que struct2 estende struct1, o que significa que se pode usar struct2. A na implementação para acessar a variável A.

```
TYPE STRUCT1 :  
STRUCT  
A: INT;  
B: BOOL;  
END_STRUCT  
END_TYPE  
  
TYPE STRUCT2 EXTENDS STRUCT1 :  
STRUCT  
C: DWORD;  
D: STRING;  
END_STRUCT  
END_TYPE
```

### Método

**NOTA:** Esta funcionalidade somente está disponível se suportada pela configuração atual.

Assim como na programação orientada a objetos, os *Métodos* (M) podem ser usados para descrever uma sequência de instruções. Assim como a função, o método não é uma POU independente, mas deve ser atribuído a um bloco funcional. Pode ser considerado como uma função que contém uma instância do respectivo bloco funcional.

### Inserindo Métodos

Para atribuir um método a um bloco funcional, selecione-o na árvore das *POUs* ou *Dispositivos*. No menu de contexto, use o comando *Acréscimo Objeto/Método*. No diálogo *Acréscimo Método*, digite o nome, o tipo de retorno e a linguagem de implementação desejados. Os tipos de dados de retorno podem ser visualizados usando o botão [...] para abrir o diálogo do *Assistente de Entrada*. O editor do método estará disponível via botão *Abrir*.

### Declaração

#### Sintaxe:

```
METHOD <NOME DO MÉTODO> : <TIPO DE DADO>  
VAR_INPUT  
X: INT;  
END_VAR
```

### Chamadas de Método

Chamadas de método são também denominadas chamadas de função virtual. Veja: **Invocação de Método**.

**NOTAS:**

- Todos dados de um método são temporários e somente são válidos durante a sua execução (variáveis de pilha).
- No corpo de um método são permitidas variáveis e o acesso à instância do bloco funcional.
- Ponteiro THIS: o identificador THIS é usado para apontar diretamente para a instância implícita do bloco funcional disponível. Observe que, em uma variável declarada localmente, pode estar escondida uma variável do bloco funcional. Então, na descrição da sintaxe do exemplo mostrado acima, THIS^.x não se refere à entrada x do método, mas sim à variável x do bloco funcional. As variáveis VAR\_IN\_OUT ou VAR\_TEMP do bloco funcional não podem ser acessadas em um método.
- Métodos, assim como funções podem ter saídas adicionais. Estas podem ser atribuídas durante a invocação do método.

### Métodos Especiais para um Bloco Funcional

- Método Init: um método `FB_Init` pode ser declarado implícita ou explicitamente. Ele contém o código de inicialização para o bloco funcional, conforme declarado na sua parte de declaração. Veja: **FB\_Init**.
- Método Reinit: se o método denominado `FB_Reinit` for declarado para uma instância de um bloco funcional, ele será chamado após esta ter sido copiada e reinicializará o novo módulo da instância. Veja: **FB\_Reinit**.
- Método Exit: se o usuário deseja um método de saída (`FB_Exit`), por exemplo, para desalocação, deve declará-lo explicitamente. Não há declaração implícita. O método de saída será chamado para cada instância do bloco funcional antes de um novo envio, reset ou durante uma alteração online para todas as instâncias movidas ou apagadas. Veja: **FB\_Exit**.
- Propriedades: para os métodos Set e Get, veja o item **Propriedade**.

### Chamada de Método (Aplicação Interrompida)

No arquivo de descrição do dispositivo, pode-se definir que um determinado método seja sempre chamado ciclicamente por uma determinada instância do bloco funcional (ou de um módulo de biblioteca). Se este método contiver os parâmetros de entrada abaixo, ele será processado também quando a aplicação corrente ativa não estiver em execução:

```
VAR_INPUT  
PTASKINFO : POINTER TO DWORD;  
PAPPLICATIONINFO: POINTER TO _IMPLICIT_APPLICATION_INFO;  
END_VAR
```


O programador pode, então, verificar o status da aplicação via `pApplicationInfo` e definir o que deve acontecer.

Exemplo:

```
IF PAPPLICATIONINFO^.STATE=RUNNING THEN <INSTRUÇÕES> END_IF
```

### Propriedade

**NOTA:** Esta funcionalidade somente está disponível se suportada pela configuração atual.

Uma *Propriedade*  é um tipo de objeto que pode ser inserido em um programa ou bloco funcional através do comando *Acrescentar Objeto* e após *Propriedade...* no menu de contexto. No diálogo *Acrescentar Propriedade*, o nome, o tipo de retorno e a linguagem de implementação desejados devem ser especificados.

Ela contém dois métodos especiais que serão inseridos automaticamente na árvore de objetos no item correspondente:

- O método Set é chamado quando a propriedade é escrita, ou seja, o nome da propriedade é usado como entrada.
- O método Get é chamado quando a propriedade é lida, ou seja, o nome da propriedade é usado como saída.

Veja o exemplo onde o bloco funcional FB1 usa uma variável local `MILLI`. Esta variável é determinada pelas propriedades Set e Get:

Código na propriedade Get:

```
SECONDS := MILLI / 1000;
```

Código na propriedade Set:

```
MILLI := SECONDS * 1000;
```

A propriedade do bloco funcional pode ser escrita (método Set), por exemplo, por `fbinst.seconds := 22;` (`fbinst` é a instância de FB1) ou pode ser lida (método Get), por exemplo, por `testvar := fbinst.seconds;`

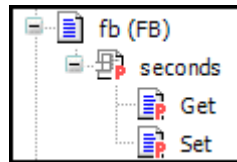



Figura 2-3. Propriedade Segundos Acrescentada ao Bloco Funcional fb

Propriedades podem ter variáveis locais adicionais, porém não tem entradas adicionais e - diferentemente da função ou método - não tem saídas adicionais.

### Monitorando uma Propriedade

As propriedades podem ser monitoradas no modo online com a ajuda da *Monitoração Online* ou da *Lista de Monitoração*. Para monitorar uma propriedade, é necessário adicionar o pragma `{attribute 'monitoring':='variable'}` no topo da sua definição.

### Ação

Ações (ícone ) podem ser definidas e atribuídas a blocos funcionais e programas através do comando *Acrescentar Objeto*. A ação é uma implementação adicional que pode ser criada em uma linguagem diferente da implementação básica. A cada ação, é referenciado um nome.

Uma ação manipula os dados do bloco funcional ou programa ao qual ela pertence. Ela utiliza as variáveis de entrada/saída e variáveis locais definidas nessas POU's e, por isso, não contém declarações próprias.

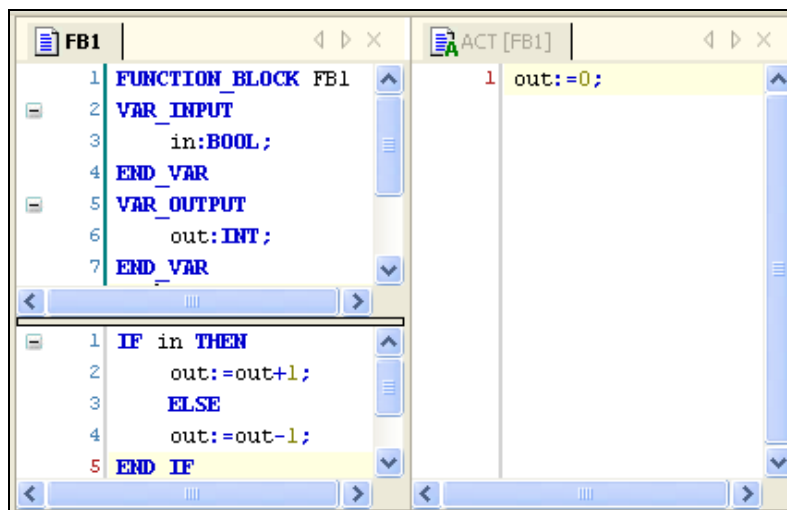


Figura 2-4. Ação em um Bloco Funcional

Neste exemplo, cada chamada do bloco funcional FB1 incrementará ou decrementará a variável de saída Out, dependendo do valor da variável de entrada In. Chamando a ação Reset deste bloco funcional, a variável de saída Out será zerada. É escrito na mesma variável Out nos dois casos.

Pode-se adicionar uma ação via comando *Acrescentar Objeto/Ação*, quando o respectivo programa ou bloco funcional for selecionado na árvore dos *Dispositivos* ou das *POUs*. No diálogo *Acrescentar Ação*, defina o nome da ação e a linguagem de implementação desejada.

### Chamando uma Ação

Uma ação é chamada com:

<NOME\_PROGRAMA>.<NOME\_AÇÃO> OU <NOME\_INSTÂNCIA>.<NOME\_AÇÃO>.

Observe as particularidades da notação em FBD (exemplo abaixo).

Para chamar uma ação dentro do seu próprio bloco (dentro do programa ou bloco funcional ao qual pertence), basta referenciar o nome da ação.

Exemplos para a chamada da ação descrita acima a partir de outra POU:

Declaração para todos os exemplos:

```
PROGRAM MAINPRG
VAR
INST : COUNTER;
END_VAR
```

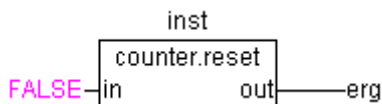
Chamada da ação Reset em outra POU programada em IL:

```
CAL INST.RESET (IN := FALSE)
LD INST.OUT
ST ERG
```

Chamada da ação Reset em outra POU programada em ST:

```
INST.RESET (IN := FALSE);
ERG := INST.OUT;
```

Chamada da ação Reset em outra POU programada em FBD:



**NOTA:** A norma IEC não reconhece outras ações que não sejam do Sequenciamento Gráfico de Funções (SFC). Nesta linguagem, as ações são parte essencial e contêm instruções a serem processadas nos passos específicos do gráfico.

### Função Externa, Bloco Funcional, Método

O programador não gerará nenhum código para uma função externa, bloco funcional ou método.

Execute os seguintes passos para criar uma POU externa:

- Adicione a POU desejada na janela das POUs do projeto, como se fosse qualquer objeto interno, e defina as respectivas variáveis de entrada e saída.

**NOTA:** Variáveis locais devem ser definidas em blocos funcionais externos, mas não podem ser definidas em funções externas ou métodos. Observe que as variáveis VAR\_STAT não podem ser usadas no sistema de execução.

No sistema de execução, deve ser implementado uma função equivalente, um bloco funcional ou um método. Em um envio de programa para cada POU externa, será pesquisada a POU equivalente no sistema de execução. Caso seja encontrada, um link será gerado.

### Lista de Variáveis Globais - GVL

Uma Lista de Variáveis Globais, a GVL (🌐) é usada para declarar variáveis globais. Se uma GVL for definida na visualização das POUs, as variáveis estarão disponíveis para todo o projeto. Se uma GVL estiver atribuída a uma determinada aplicação, as variáveis serão válidas dentro desta aplicação.


Uma GVL pode ser adicionada através do comando *Acréscimo Objeto* e *Acréscimo Lista de Variáveis Globais*. Para atribuí-la a uma aplicação existente, escolha o comando no menu de contexto enquanto uma aplicação é selecionada na árvore de dispositivos. Caso contrário, a nova GVL será adicionada à visualização das POUs.

O editor GVL é usado para editar uma Lista de Variáveis Globais.

Se o dispositivo suportar a funcionalidade de rede, as variáveis contidas em uma GVL podem ser configuradas para estarem disponíveis como variáveis de rede, isto é, para uma troca de dados via broadcast com outros dispositivos na rede. Para tanto, as propriedades de rede apropriadas devem estar configuradas para a GVL.

Considere que as variáveis declaradas nas GVLs sempre são inicializadas antes das variáveis locais das POU's.

### Variáveis Persistentes

O objeto  representa uma lista de variáveis globais contendo apenas as variáveis persistentes da aplicação, e pode ser atribuído a uma aplicação através da inserção do mesmo na árvore de dispositivos via comando *Acréscitar Objeto*.

Variáveis remanentes são reinicializadas somente em um *Reset Origem*. Para informações adicionais, consulte o item **Variáveis Remanentes**.

A lista de variáveis persistentes é editada no editor GVL e VAR\_GLOBAL PERSISTENT RETAIN aparece na primeira linha.

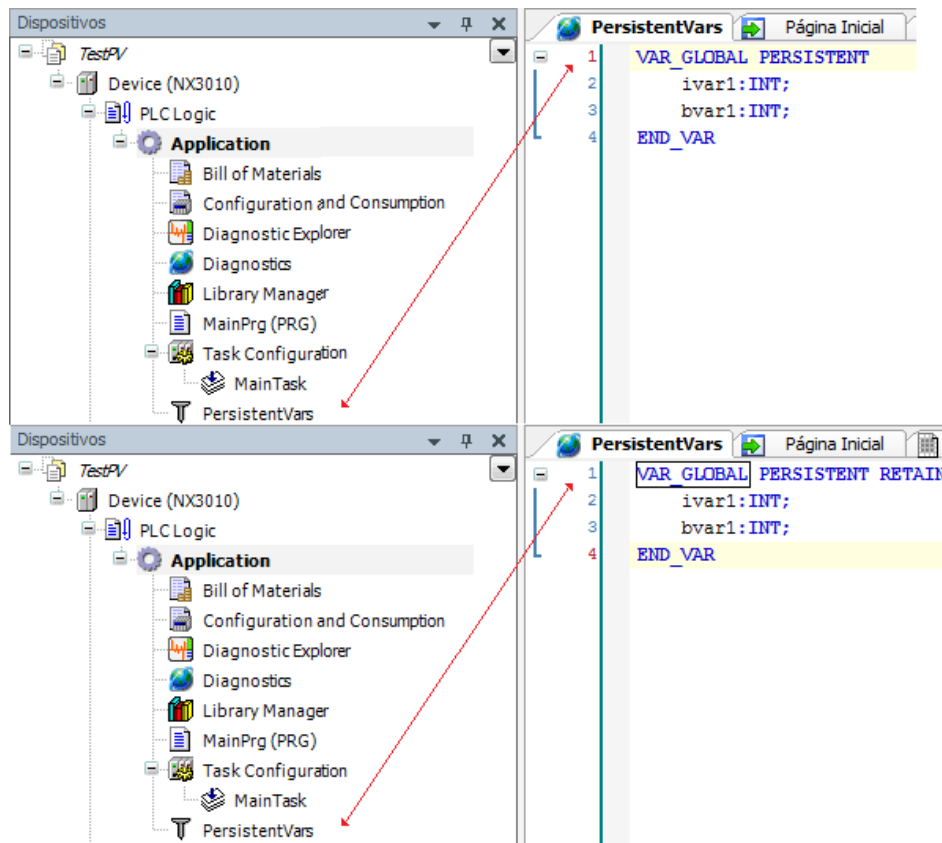



Figura 2-5. Lista de Variáveis Persistentes

### Arquivo Externo

Qualquer arquivo externo (com qualquer extensão) pode ser adicionado ao projeto na janela das POU's através do comando *Acréscitar Objeto*, opção *Arquivo Externo*. O diálogo *Acréscitar Arquivo Externo* será aberto.

Pressione o botão  para navegar para um arquivo, o caminho que será inserido no campo *Caminho do arquivo*. No campo *Nome* automaticamente o nome do arquivo escolhido será inserido

sem extensão. Você pode editar este campo para definir um outro nome para o arquivo dentro do projeto.


Seguintes opções podem ser selecionadas em relação ao arquivo externo.

- *Lembrar link*: O arquivo estará disponível no projeto somente se ele está disponível no caminho definido.
- *Lembrar link e incluir no projeto*: uma cópia do arquivo será armazenado internamente no projeto, mas também o link para o arquivo externo será lembrado. Com esta opção for marcada uma das opções abaixo deve ser escolhida:
  - *Recarrega o arquivo automaticamente*: O arquivo será atualizado dentro do projeto assim que ele foi alterado externamente.
  - *Solicita para recarregar o arquivo*: Uma caixa de diálogo irá aparecer assim que o arquivo foi alterado externamente. Você pode decidir se o arquivo deve ser atualizado também dentro do projeto.
  - *Nenhuma ação*: O arquivo permanecerá inalterada, mesmo quando ele for alterado externamente.
- *Incluir no projeto*: uma cópia do arquivo será armazenado no projeto. Não haverá nenhuma conexão adicional para o arquivo fora do projeto.

No diálogo ainda existe o botão *Exibir Propriedades dos Arquivos...*, ele abre o diálogo padrão para as propriedades, mas com uma aba adicional *Arquivo Externo*, onde as propriedades que foram definidas na caixa de diálogo *Acrescentar Arquivo Externo* podem ser visualizadas e modificadas.

### POUs para Verificações Implícitas

Caso o usuário deseje utilizar a funcionalidade de verificação de ARRAYS, limites de intervalo, divisões por zero e ponteiros durante a execução, POU's especiais poderão ser adicionadas à aplicação. Esta funcionalidade está implícita.

Para tanto, o menu *Acrescentar Objeto* na categoria *POUs para Verificações Implícitas*  apresenta as seguintes funções:

- CheckBounds
- CheckDivDInt
- CheckDivLInt
- CheckDivReal
- CheckDivLReal
- CheckRangeSigned
- CheckRangeUnsigned
- CheckPointer

Após ter sido inserida uma POU de verificação, a mesma será aberta no editor correspondente na linguagem de implementação selecionada. No editor ST, está disponível uma implementação padrão adaptável aos requisitos do usuário.

A opção de inserção de uma determinada POU de verificação só está disponível uma vez. Se todos os tipos de POU de verificação já tiverem sido adicionados na aplicação, o diálogo *Acrescentar Objeto* não mais apresentará a categoria *POUs para Verificações Implícitas*.


NOTA: Para manter a funcionalidade de verificação, não modifique a parte de declaração de uma função de verificação implícita.
--

### Gerenciamento de Bibliotecas

Bibliotecas podem fornecer funções, blocos funcionais, tipos de dados, variáveis globais e até mesmo visualizações, que podem ser usadas em um projeto assim como em outras POU's e variáveis definidas diretamente no projeto.

A extensão padrão para um arquivo de biblioteca é `.library`.



O gerenciamento das bibliotecas em um projeto é feito no *Library Manager* () , cuja instalação no sistema é feita previamente no diálogo do *Repositório de Bibliotecas*.

NOTA: O diálogo *Repositório de Bibliotecas* só está disponível se as características pré-definidas escolhidas pelo usuário for *Profissional* ou se a opção *Habilitar Diálogo de Repositório* estiver habilitado. Para mais informações sobre características, consulte **Características** no Manual de Utilização MasterTool IEC XE – MU299048.

As funções do projeto para pesquisa local/global e substituição também são aplicáveis às bibliotecas incluídas.

Consulte informações gerais sobre:

- Instalação e inclusão no projeto
- Bibliotecas referenciadas
- Versões de bibliotecas
- Acesso exclusivo aos módulos de bibliotecas ou variáveis
- Criando bibliotecas no MasterTool IEC XE, codificação e documentação
- Módulos de biblioteca internos e externos

### Instalação e Inclusão no Projeto

- Bibliotecas podem ser gerenciadas no sistema local em um ou vários repositórios (pastas, locais), elas não podem ser incluídas no projeto antes de ter sido instalada em um repositório no sistema local. A instalação é feita no diálogo *Repositório de Bibliotecas*.
- Como pré-requisito para a instalação, uma biblioteca deve conter título, versão e nome da empresa nas suas *Informações do Projeto*. Opcionalmente, pode ser definida também uma categoria, o que será útil posteriormente para a classificação destas no gerenciador.
- Se a categoria não estiver definida nas *Informações do Projeto*, a biblioteca será automaticamente considerada como pertencente à categoria *Miscelânea*. Outras categorias podem ser definidas em um ou mais arquivos xml \*.libcat.xml, os quais também são carregados no diálogo *Informações de Projeto* para selecionar uma das categorias. Consulte abaixo **Criando Bibliotecas, Codificação, Documentação**.
- O *Library Manager* é usado para incluir bibliotecas em um projeto. Em um *Projeto MasterTool Padrão*, o *Library Manager* está automaticamente adicionado na janela de visualização dos *Dispositivos* ou *POUs*, através do diálogo *Acrescentar Objeto*. Bibliotecas referenciadas em outras bibliotecas também são exibidas no gerenciador, incluindo as bibliotecas ocultas. Consulte abaixo: **Bibliotecas Referenciadas**.
- Se o arquivo \*.library está disponível por inteiro, isto é, não apenas a sua versão compilada (\*.compiled-library), as POU's da referida biblioteca podem ser abertas com um duplo clique nas suas referências, dentro do gerenciador.
- Quando um módulo da biblioteca é chamado por uma aplicação, a pesquisa será feita em todas as bibliotecas e repositórios na ordem definida no diálogo *Repositório de Bibliotecas*.

### Bibliotecas Referenciadas

- As bibliotecas podem incluir outras bibliotecas referenciadas (aninhamento). Ao acrescentar uma biblioteca que inclua outras no gerenciador, as bibliotecas referenciadas serão acrescentadas automaticamente.
- O comportamento que a biblioteca referenciada apresentará ao ser inserida em um projeto com sua biblioteca principal pode ser definido nas suas *Propriedades*.
- A visibilidade da biblioteca referenciada no *Library Manager* pode ser desativada. Elas passam a ter o status de bibliotecas ocultas.
- Bibliotecas do tipo container são aquelas sem módulos próprios e que apenas referenciam outras bibliotecas. A biblioteca container possibilita criar um projeto com várias bibliotecas incluídas ao mesmo tempo. O acesso aos seus módulos pode ser simplificado tornando a biblioteca de nível superior (topo), pois isto permite omitir o contexto desta biblioteca no caminho do acesso. Para

tanto, basta ativar o botão *Publicar...* , mas esta opção somente deve ser ativada na biblioteca container e deve ser gerenciada com muito cuidado.

### Versões de Biblioteca

- O sistema permite a instalação de várias versões da mesma biblioteca.
- Nos projetos com múltiplas versões de bibliotecas, a escolha de qual será usada na aplicação segue o seguinte critério:
  - No mesmo nível dentro do mesmo gerenciador, a escolha da versão (uma determinada ou a mais recente), dependerá das propriedades da biblioteca.
  - Em níveis diferentes dentro do mesmo gerenciador (bibliotecas referenciadas), o acesso único aos módulos da biblioteca ou às variáveis será possível acrescentando-se o contexto adequado (veja no parágrafo a seguir).

### Acesso Único aos Módulos da Biblioteca ou às Variáveis

- Basicamente, se houver vários módulos ou variáveis com o mesmo nome no projeto, o acesso a eles deve ser único, caso contrário, serão gerados erros de compilação. Isto se aplica aos módulos de projetos locais ou variáveis e também aos elementos de software disponíveis nas bibliotecas incluídas e suas referenciadas. Para conferir exclusividade nestes casos, defina um prefixo para o nome do módulo através do apropriado contexto da biblioteca.
- O contexto padrão da biblioteca é definido nas propriedades das bibliotecas. Caso não seja definido explicitamente, o contexto será igual ao nome da biblioteca. O contexto padrão também pode ser especificado no diálogo *Propriedades*. Após a inclusão da biblioteca no projeto, o contexto pode ser alterado localmente pelo usuário (no diálogo *Propriedades*).

Exemplos: suponha que o contexto da biblioteca Lib1 é especificado nas propriedades para ser Lib1. Veja na coluna da direita o uso dos contextos para acesso único à variável var1 definida nos módulos module1 e POU1.

	Variável var1 está disponível nos seguintes locais...	Acesso a var1 usando o caminho do contexto apropriado
1	Na biblioteca Lib1 no Gerenciador de bibliotecas global na janela das POU's.	Lib1.module1.var1
2	Na biblioteca Lib1 no Gerenciador de bibliotecas na Aplicação App1 do Dispositivo Dev1 na janela dos dispositivos.	Dev1.App1.Lib1.module1.var1
3	Na biblioteca Lib1 incluída em F_Lib no Gerenciador de bibliotecas global na janela das POU's.	Por padrão (a opção Publicar... nas Propriedades da biblioteca está desativada): F_Lib.Lib1.module1.var1 Se a opção Publicar... estivesse ativada, module1 seria tratado com um componente de uma biblioteca de nível superior. Assim, o acesso somente seria possível através de Lib1.module1.var1 ou module1.var1 . Neste exemplo, entretanto, isto poderia provocar erros de compilação, já que o caminho de acesso não é exclusivo, veja (1) e (4).
4	No objeto module1 definido na janela das POU's.	module1.var1
5	No objeto POU1 definido na janela das POU's.	POU1.var1

Tabela 2-1. Acesso Único aos Módulos da Biblioteca ou às Variáveis

### Criando Bibliotecas, Codificação, Documentação

Um projeto do MasterTool IEC XE pode ser salvo como uma biblioteca (<nome do projeto>.library) e, opcionalmente, pode ser instalado no Repositório de Bibliotecas do Sistema. Nos projetos de biblioteca, somente serão considerados os objetos gerenciados na janela das POU's. Para estes projetos, é recomendado escolher o modelo *Biblioteca Vazia* no diálogo *Novo Projeto*. Observe o que segue:

- Nas *Informações do Projeto*, devem estar especificados um título, uma versão e a empresa. A alteração do contexto padrão pode ser feita, então, se necessário. Opcionalmente, recomenda-se acrescentar uma categoria, pois ela será útil posteriormente para a classificação das bibliotecas no *Library Manager* e no *Repositório de Bibliotecas*. Se ela for diferente da padrão (Miscelânea), deve ser carregada uma descrição da categoria (arquivo XML \*.libcat.xml ou outra biblioteca que contenha este arquivo de descrição). Caso seja necessário, pode ser criado um novo arquivo de descrição da categoria ou um arquivo existente pode ser modificado. A informação das categorias selecionadas e o arquivo básico de descrição da categoria serão transferidos para o projeto local e posteriormente - ao instalar a biblioteca - para o repositório específico. A partir daí, as bibliotecas estarão categorizadas no repositório. Caso outra biblioteca apresente um arquivo de descrição com o mesmo ID, mas com conteúdo diferente do arquivo novo, ela será válida no repositório.
- Se a biblioteca incluir outras bibliotecas, atente para o comportamento das bibliotecas referenciadas quando a principal for incluída no projeto. Isto se refere ao tratamento da versão, contexto, visibilidade e propriedades de acesso (configurados no diálogo propriedades da biblioteca referenciada). Para que ela possa, posteriormente, referenciar outra (biblioteca específica do dispositivo), um espaço reservado pode ser usado ao configurar a referência. Para que os módulos da biblioteca possam ser protegidos contra visualização e acesso, o projeto pode ser salvo em um formato codificado (<nome do projeto>.compiled-library).
- As estruturas de dados de uma biblioteca são internas. Este objetos ficam ocultos e, portanto, não aparecem no gerenciador, assim como também não aparece a funcionalidade *Componentes da lista* nem o *Assistente de Entrada*.
- Para que o usuário possa acessar facilmente as informações sobre um módulo de biblioteca, um comentário pode ser adicionado à declaração do parâmetro do módulo. Este comentário será

exibido posteriormente quando a biblioteca for incluída no projeto, na guia *Documentação* do *Gerenciador de Bibliotecas*.

- Os comandos a seguir estão disponíveis por padrão no menu *Arquivo* para salvar um projeto de biblioteca:
  - *Salvar Projeto Como...*
  - *Salvar Projeto Como Biblioteca Compilada*
  - *Salvar Projeto e Instalar no Repositório de Bibliotecas*

## 3. Comandos do Menu

Neste capítulo serão tratados apenas comandos dos menus que fazem o gerenciamento de bibliotecas, para consultar comandos de outros menus consulte **Comandos de Menu** no Manual de Utilização MasterTool IEC XE – MU299048.

### Library Manager

Esta categoria fornece os comandos do editor *Library Manager* para o tratamento das bibliotecas a serem usadas no projeto.

#### Comandos do Gerenciador de Bibliotecas

Fornece os comandos listados abaixo. Eles fazem parte do menu *Bibliotecas* quando o *Library Manager* esta ativo.

Comandos disponíveis:

- Acrescentar Biblioteca...
- Propriedades...
- Tentar Recarregar a Biblioteca

Para obter informações gerais sobre o gerenciamento de bibliotecas no MasterTool IEC XE, consulte **Library Manager**.

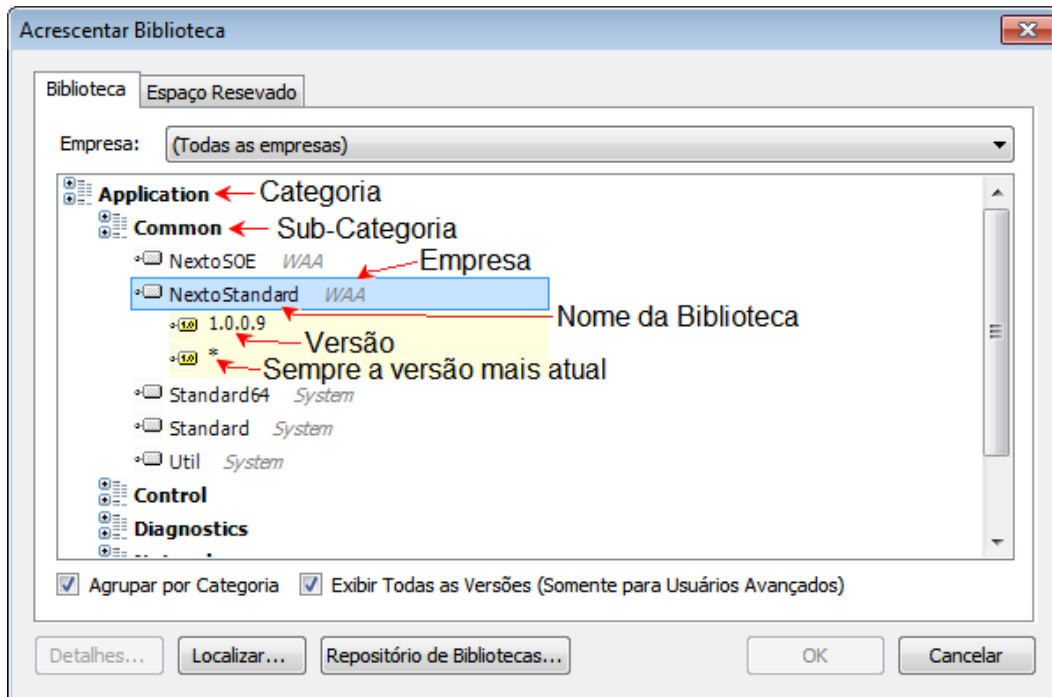
#### Acrescentar Biblioteca

Este comando faz parte do menu *Bibliotecas* e da janela do editor *Library Manager*.

Utiliza-se este comando para incluir bibliotecas no projeto. Somente podem ser adicionadas bibliotecas que já estão instaladas no sistema. É possível incluir múltiplas versões de uma biblioteca ao mesmo tempo dentro de um gerenciador de bibliotecas.

O comando abre o diálogo *Acrescentar Biblioteca* e apresenta as guias *Biblioteca* e *Espaço Reservado* (*Espaço Reservado* disponível somente com a característica *Habilitar Diálogo do Repositório* selecionada, para mais informações consulte o item **Características** no Manual de Utilização MasterTool IEC XE – MU299048).

## Subdiálogo Biblioteca



**Figura 3-1. Diálogo Acrescentar Biblioteca (Todas as Versões)**

Aqui, todas as bibliotecas atualmente instaladas no sistema serão listadas. A exibição pode ser filtrada definindo-se uma determinada *Empresa* a partir da lista de seleção. A opção *Todas as Empresas* listará todas as bibliotecas disponíveis.

Se a opção *Agrupar por Categoria* estiver ativada, as bibliotecas da empresa atualmente definida serão listadas de acordo com as categorias disponíveis, caso contrário, a lista estará em ordem alfabética. Se esta opção for escolhida, as categorias aparecerão sob a forma de nós e as bibliotecas ou categorias adicionais serão exibidas abaixo destes nós (em relação à categoria atualmente selecionada).

O usuário, então, escolhe a biblioteca desejada. Se a opção *Exibir Todas as Versões (Somente para Usuários Avançados)* estiver ativada, todas as versões instaladas desta biblioteca aparecerão recuadas sob ela.

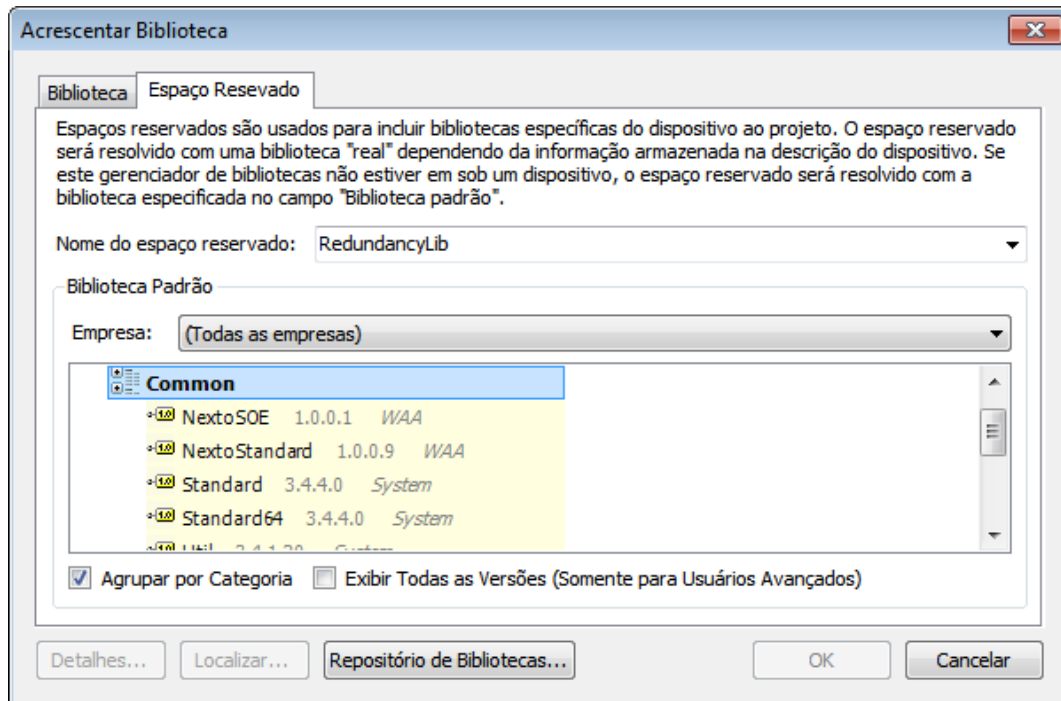
Além dos identificadores explícitos da versão, está disponível um “\*”, sinal que significa “última versão”. Assim, é possível escolher uma entre as versões. Por padrão, entretanto, esta opção está desativada e somente a última versão é exibida. Também é possível realizar uma multiseleção de bibliotecas: basta manter a tecla <SHIFT> pressionada ao mesmo tempo em que se realiza a seleção das mesmas.

Após a confirmação da seleção (via *OK*), as bibliotecas são adicionadas à lista na janela do *Library Manager*.

Para incluir uma biblioteca ainda não instalada no sistema local, usa-se o botão *Repositório de Bibliotecas*, o qual abre o diálogo que permite realizar a instalação.

NOTA: O diálogo *Repositório de Bibliotecas* só está disponível se as características pré-definidas escolhidas pelo usuário for *Profissional* ou se a opção *Habilitar Diálogo de Repositório* estiver habilitado. Para mais informações sobre características, consulte **Características** no Manual de Utilização MasterTool IEC XE – MU299048.

## Subdiálogo Espaço Reservado



**Figura 3-2. Diálogo Acrescentar Biblioteca, Espaço Reservado**

NOTA: *Espaço Reservado* só está disponível se as características pré-definidas escolhidas pelo usuário for *Profissional* ou se a opção *Habilitar Diálogo de Repositório* estiver habilitado. Para mais informações sobre características, consulte **Características** no Manual de Utilização MasterTool IEC XE – MU299048.

A guia *Espaço Reservado* é usada nos dois casos seguintes:

- Criação de um projeto independente de dispositivo
- Criação de um projeto de biblioteca <library\_xy>, que referencia outra biblioteca destinada a ser específica do dispositivo

### Espaço Reservado no Projeto

Para que um projeto seja compatível com vários dispositivos intercambiáveis, as bibliotecas específicas dos dispositivos devem ser incluídas no gerenciador de bibliotecas do projeto através de espaços reservados.

Assim que o dispositivo for especificado, os espaços reservados serão selecionados de acordo com a descrição deste dispositivo. Mesmo se não houver uma descrição de dispositivo disponível, os espaços reservados permitirão que o projeto passe por uma verificação sintática.

Para incluir uma biblioteca no gerenciador através de um espaço reservado, esta deve ser selecionada no campo *Biblioteca Padrão*. Assim é possível catalogar as bibliotecas propostas de acordo com a empresa fornecedora.

Além disto, o nome do espaço reservado deve ser inserido no campo de edição associado. Para garantir a correta inserção do nome, usa-se a lista de seleção que oferece todos os nomes dos espaços reservados atualmente definidos nas descrições dos dispositivos.

### Espaço Reservado no Projeto da Biblioteca

Se o projeto estiver baseado em outras bibliotecas que não as específicas do dispositivo, estas deverão ser incluídas através de espaços reservados.

Isto significa que, em vez de especificar uma biblioteca em particular a ser incluída, será inserido um espaço reservado, o qual, posteriormente, será substituído. Quando a <biblioteca\_xy> for usada em outro projeto para um determinado dispositivo, por exemplo, este espaço reservado será substituído pelo nome de uma biblioteca definida especificamente pelo dispositivo. Este nome deve ser especificado no respectivo arquivo de descrição do dispositivo (<biblioteca\_xy>) que atribui o nome do espaço reservado ao nome verdadeiro da biblioteca.

Se o gerenciador, por qualquer razão, não estiver atribuído ao dispositivo, o espaço reservado será substituído pela biblioteca padrão especificada neste diálogo. Isto permite, por exemplo, que a compilação do projeto da biblioteca seja editada sem erros, mesmo que não haja uma descrição adequada do dispositivo disponível no momento.

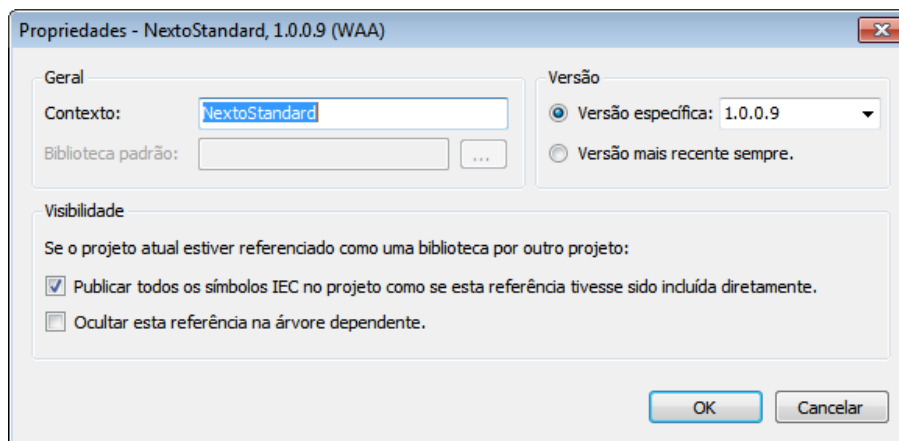
No campo *Nome do Espaço Reservado*, digita-se qualquer string como um nome para este espaço reservado. Após, escolhe-se uma biblioteca padrão a partir das bibliotecas atualmente instaladas. Isto deve ser feito conforme a descrição acima para acrescentar uma biblioteca no subdiálogo *Bibliotecas*. A opção *Exibir Todas as Versões (Somente para Usuários Avançados)* deve estar ativada para que sejam exibidas todas as versões das bibliotecas atualmente instaladas.

Após fechar o diálogo com *OK*, a biblioteca de espaço reservado será inserida na árvore do gerenciador. Ao abrir o diálogo *Propriedades* para o espaço reservado da biblioteca, o usuário obterá informações sobre a biblioteca padrão atualmente instalada.

## Propriedades

Este comando está disponível na janela do editor *Library Manager*.

Ele abre o diálogo *Propriedades* para a biblioteca atualmente selecionada na janela do gerenciador e permite algumas configurações referentes aos espaços reservados, tratamento de versão, disponibilidade e visibilidade das bibliotecas referenciadas.



**Figura 3-3. Diálogo Propriedades para Bibliotecas**

- **Contexto:** exibe o contexto atual da biblioteca. Em geral, o contexto de uma biblioteca é idêntico ao seu nome, a não ser que outra string tenha sido explicitamente definida nas informações do projeto (durante a criação do projeto da biblioteca). É possível editar o contexto a qualquer momento neste diálogo. Para obter mais informações sobre contextos de bibliotecas, consulte o item **Editor Library Manager** no Manual de Utilização MasterTool IEC XE – MU299048.
- **Biblioteca padrão:** (disponível somente com a característica *Habilitar Diálogo do Repositório* selecionada, consulte o item **Características** no Manual de Utilização MasterTool IEC XE – MU299048) se o espaço reservado de uma biblioteca está selecionado no gerenciador, este campo mostra o nome da biblioteca que deve substituir o espaço reservado (se não houver nenhuma biblioteca do dispositivo disponível). Consulte acima o item correspondente para obter mais informações sobre espaços reservados.



- **Versão:** (disponível somente com a característica *Habilitar Diálogo do Repositório* selecionada, consulte o item **Características** no Manual de Utilização MasterTool IEC XE – MU299048) define-se qual versão da biblioteca deve ser usada no projeto:
  - **Versão específica:** será usada exatamente a versão digitada (a partir da lista de seleção).
  - **Versão mais recente sempre:** será usada sempre a versão mais recente encontrada no repositório de bibliotecas, ou seja, os módulos atualmente usados podem ser alterados caso seja disponibilizada uma versão mais nova.
- **Visibilidade:** (disponível somente com a característica *Habilitar Diálogo do Repositório* selecionada, consulte o item **Características** no Manual de Utilização MasterTool IEC XE – MU299048) estas configurações são importantes quando a biblioteca é incluída (referenciada por outra biblioteca). Por padrão esta opção não está ativa:
  - **Publicar todos os símbolos IEC no projeto como se esta referência tivesse sido incluída diretamente:** enquanto esta opção estiver desativada, os componentes da biblioteca atual - se referenciada por outra - poderão ser acessados exclusivamente através do caminho adequado do espaço reservado (composto dos espaços reservados da biblioteca principal e do seu próprio espaço reservado, além dos módulos e do identificador da variável).
  - **Ocultar esta referência na árvore de dependência:** se esta opção estiver ativada, a biblioteca atual não será exibida quando a biblioteca principal for incluída no projeto. Isto permite incluir bibliotecas ocultas, mas é recomendado que o usuário tenha cuidado ao usar esta opção, pois em caso de mensagens de erro, pode haver dificuldade em localizar a biblioteca original.

NOTA: A opção *Publicar todos os símbolos IEC no projeto como se esta referência tivesse sido incluída diretamente* somente deve ser ativada se o usuário desejar usar bibliotecas container (sem módulos próprios) e incluir outras bibliotecas com o objetivo de compactá-las. Esta compactação, permite, por exemplo, incluir de uma só vez várias bibliotecas no projeto. Neste caso, entretanto, é desejável que as bibliotecas específicas estejam no nível superior do gerenciador de bibliotecas para que os módulos possam ser acessados diretamente e o espaço reservado da biblioteca container possa ser deixado de fora.

#### Tentar Recarregar a Biblioteca

Este comando faz parte do menu *Bibliotecas* e da janela do editor *Library Manager*.

Se uma biblioteca incluída em um projeto, por alguma razão, não estiver disponível no caminho definido na abertura do projeto no programador, será gerada uma mensagem apropriada. O usuário deve verificar os erros, disponibilizar a biblioteca corretamente e usar o comando *Tentar Recarregar a Biblioteca* (enquanto a biblioteca estiver selecionada no gerenciador). Desta forma, a biblioteca pode ser recarregada sem a necessidade de sair do projeto.

## 4. Referência de Programação

### Declaração

As variáveis de um projeto devem ser declaradas manualmente no editor de declaração ou através do diálogo *Autodeclarar*. Consulte os itens relacionados como, por exemplo, categorias de variáveis (locais, globais, entrada, saída, etc.), inicialização, pragmas, método init, entre outros.

### Declaração de Variáveis

A declaração de uma variável pode ser realizada na parte de declaração de uma POU ou via diálogo *Autodeclarar*, assim como em uma DUT ou editor GVL.

A classe (ou escopo, no diálogo de declaração) da variável a ser declarada é especificada pelas palavras-chave no âmbito de sua parte de declaração. A declaração de uma variável comum, por exemplo, aparece entre as palavras VAR e END\_VAR.

```
VAR_INPUT
VAR_OUTPUT
VAR_IN_OUT
VAR_GLOBAL
VAR_TEMP
VAR_STAT
VAR_EXTERNAL
VAR_CONFIG
```

As palavras-chave do tipo da variável devem estar complementadas pelas palavras-chave de atributo, como por exemplo RETAIN (VAR\_INPUT RETAIN).

A declaração de uma variável deve atender a determinadas regras.

Sintaxe:

```
<IDENTIFICADOR> {AT <ENDEREÇO>} :<TIPO> {:=<INICIALIZAÇÃO>};
```

As partes entre chaves ({} ) são opcionais.

O identificador é o nome da variável. Os itens listados a seguir, em cada caso, devem ser considerados ao definir um identificador. Veja o item **Recomendações na Nomeação de Identificadores**.

- Ele não deve conter espaços nem caracteres especiais
- Ele não diferencia maiúsculas e minúsculas, o que significa que VAR1, Var e var1 são todas a mesma variável
- O caractere sublinhado (por exemplo, A\_BCD e AB\_CD são considerados diferentes identificadores), porém um identificador não deve conter mais de um sublinhado por linha
- O comprimento do identificador (identificação) é ilimitado
- As regras abaixo se referem ao múltiplo uso de identificadores

#### Uso Múltiplo de Identificadores (Contextos)

Um identificador não deve ser usado duplamente no modo local e não pode ser idêntico a uma palavra-chave.

Globalmente, um identificador pode ser usado mais de uma vez. Assim, uma variável local pode ter o mesmo nome de uma variável global. Em uma POU, neste caso, a variável local terá prioridade.

Uma variável definida em uma lista de variáveis globais pode ter o mesmo nome de uma variável definida em outra lista de variáveis globais. Neste contexto, observe as extensões da norma IEC 61131-3:

- Operador de escopo global: um caminho de instância iniciando com “.” abre um escopo global. Assim, se houver uma variável local, por exemplo, `ivar`, com o mesmo nome de uma variável global, `.ivar` está referindo-se à variável global.
- O nome de uma lista de variáveis globais pode ser usado como um contexto para as variáveis incluídas. Assim, as variáveis podem ser declaradas com o mesmo nome em diferentes listas e podem ser acessadas precedendo o nome desta lista antes do nome da variável. Exemplo:

```
GLOBLIST1.IVAR := GLOBLIST2.IVAR; (* IVAR da GLOBLIST2 é copiado para IVAR na GLOBLIST1 *)
```

- Variáveis definidas em uma lista de variáveis globais de uma biblioteca incluída podem ser acessadas de acordo com a sintaxe <Contexto da biblioteca>.<Nome da Lista de Variáveis Globais>.<Variável>. Veja abaixo informações sobre contextos de bibliotecas. Exemplo:

```
GLOBLIST1.IVAR := LIB1.GLOBLIST1.IVAR (* IVAR da GLOBLIST1 na biblioteca LIB1 é copiada para IVAR na GLOBLIST1 *)
```

No caso de uma biblioteca, um contexto também é definido quando esta for incluída através do *Library Manager*. Assim, um módulo de biblioteca ou variável pode se acessado através de <Contexto da biblioteca>. <Nome do módulo | Nome da variável>. No caso de bibliotecas aninhadas, os contextos destas devem ser indicados sucessivamente. Se `Lib1` foi referenciado por `Lib0`, o módulo `fun`, como parte de `Lib1` é acessado por `Lib0.Lib1.fun`:

```
IVAR := LIB0.LIB1.FUN(4, 5); (* valor de retorno de FUN é copiado para a variável IVAR no projeto *)
```

A ativação da caixa *Publicar todos os símbolos IEC para o projeto*, como se esta referência tivesse sido incluída ali diretamente, nas propriedades da biblioteca referenciada `Lib` faz com que o módulo `fun` também possa ser acessado diretamente via `Lib0.fun`.

### AT <Endereço>

A variável pode ser diretamente vinculada a um endereço definido usando a palavra-chave `AT`.

Nos blocos funcionais, variáveis também podem ser especificadas com declarações de endereços incompletos. Para que cada variável possa ser usada em uma instância local, deve haver um item para ela na configuração de variáveis.

### Tipo

Tipo de dado válido, opcionalmente complementado por “:=<Inicialização>”.

Opcionalmente, podem ser adicionadas instruções de pragma na parte de declaração de um objeto, para modificar a geração de código.

NOTA: Atente para a possibilidade de uma declaração automática. Para entradas mais rápidas das declarações, use o modo de atalho.
---

### Recomendações na Nomeação de Identificadores

Identificadores são definidos na declaração de variáveis (nomes das variáveis), tipos de dados definidos pelo usuário e na criação de POUs (funções, blocos funcionais e programas). Além das questões usualmente consideradas na definição de um identificador, alguns itens devem ser observados no intuito de nomeá-los o mais exclusivamente possível.

### Nomes de Variáveis

A nomeação das variáveis nas aplicações, dentro do possível, deve seguir a notação húngara.

Para cada variável, deve haver um nome de base, isto é, uma descrição breve, porém significativa.

A primeira letra de cada palavra de um nome de base deve ter uma letra maiúscula. As demais devem ser minúsculas (exemplo: `FileSize`).

Antes do nome de base, são adicionados prefixos em letras minúsculas correspondentes ao tipo de dado da variável.

Veja, na tabela abaixo, algumas informações e os prefixos recomendados para os tipos de dados específicos:

Tipo de dado	Limite inferior	Limite superior	Conteúdo da informação	Prefixo	Comentário
BIT	0	1	1 Bit	b	
BOOL	FALSE	TRUE	1 Bit	x	
BYTE			8 Bits	by	Bit string, não para operações aritméticas
WORD			16 Bits	w	Bit string, não para operações aritméticas
DWORD			32 Bits	dw	Bit string, não para operações aritméticas
LWORD			64 Bits	lw	Não para operações aritméticas
SINT	-128	127	8 Bits	si	
USINT	0	255	8 Bits	usi	
INT	-32.768	32.767	16 Bits	i	
UINT	0	65.535	16 Bits	ui	
DINT	-2.147.483.648	2.147.483.647	32 Bits	di	
UDINT	0	4.294.967.295	32 Bits	udi	
LINT	$-2^{63}$	$2^{63} - 1$	64 Bits	li	
ULINT	0	$2^{64} - 1$	64 Bits	uli	
REAL			32 Bits	r	
LREAL			64 Bits	lr	
STRING				s	
TIME				tim	
TIME_OF_DAY				tod	
DATE_AND_TIME				dt	
DATE				date	
ENUM			16 Bit	e	
POINTER				p	
ARRAY				a	

**Tabela 4-1. Tipos de Dados**

**Nota:**

**x:** Especificamente para variáveis Booleanas, o prefixo escolhido é o “x” (para diferenciar de BYTE e para adaptar-se à percepção do programador IEC). Veja endereçamento %IX0.0.

Exemplos:

```
bySubIndex: BYTE;
sFileName: STRING;
udiCounter: UDINT;
```

Nas declarações aninhadas, os prefixos são anexados uns aos outros, na ordem das mesmas.

Exemplo:

```
pabyTelegramData: POINTER TO ARRAY [0..7] OF BYTE;
```

Instâncias de blocos funcionais e variáveis de tipos de dados definidos pelo usuário, assim como um prefixo, obtêm um atalho para o nome do tipo de dado FB (por exemplo: sdo).

### Exemplo:

```
cansdoReceivedTelegram: CAN_SDOTelegram;
TYPE CAN_SDOTelegram : (* Prefixo: sdo *)
STRUCT
wIndex:WORD;
bySubIndex:BYTE;
byLen:BYTE;
aby: ARRAY [0..3] OF BYTE;
END_STRUCT
END_TYPE
```

As constantes locais (c) iniciam com o prefixo “c”, seguido de um sublinhado e do prefixo do tipo e do nome da variável.

### Exemplo:

```
VAR CONSTANT
c_uiSyncID: UINT := 16#80;
END_VAR
```

Para variáveis globais (g) e constantes globais (gc), um prefixo adicional sublinhado será anexado ao prefixo da biblioteca.

### Exemplos:

```
VAR_GLOBAL
CAN_g_iTest: INT;
END_VAR
VAR_GLOBAL CONSTANT
CAN_gc_dwExample: DWORD;
END_VAR
```

### *Nomes de Variáveis nas Bibliotecas do MasterTool IEC XE*

Basicamente, veja acima **Nomes de Variáveis**, com a seguinte exceção: variáveis globais e constantes não precisam de um prefixo de biblioteca, pois os contextos fazem as vezes do prefixo.

### Exemplo:

```
g_iTest: INT; (* Declaração *)
CAN.g_iTest (* Implementação: chamada em um programa aplicativo *)
```

### *Tipos de Dados Definidos pelo Usuário (DUT)*

Estrutura: o nome de cada tipo de dado de estrutura consiste em um prefixo de biblioteca (exemplo: COM), um sublinhado e uma breve descrição da estrutura (exemplo: SDOTELEGRAM). O prefixo associado para as variáveis usadas desta estrutura devem vir diretamente após a vírgula.

### Exemplo:

```
TYPE COM_SDOTelegram : (* Prefixo: sdo *)
STRUCT
wIndex:WORD;
bySubIndex:BYTE;
byLen:BYTE;
abyData: ARRAY [0..3] OF BYTE;
END_STRUCT
END_TYPE
```

As enumerações iniciam com o prefixo da biblioteca (exemplo: CAL), seguido de um sublinhado e do identificador em letras maiúsculas.

ENUMs devem ser definidos com os valores INT corretos.

Exemplo:

```
TYPE CAL_Day : (  
CAL_MONDAY,  
CAL_TUESDAY,  
CAL_WEDNESDAY,  
CAL_THIRSDAY,  
CAL_FRIDAY,  
CAL_SATURDAY,  
CAL_SUNDAY) ;
```

Declaração:

```
eToday: CAL_Day;
```

### *Tipos de Dados Definidos pelo Usuário (DUTs) nas Bibliotecas do MasterTool IEC XE*

Tipos de dados definidos pelo usuário (DUTs) nas bibliotecas do MasterTool IEC XE não precisam de um prefixo de biblioteca, pois os contextos fazem as vezes deste. Os componentes da enumeração também são definidos sem prefixos de biblioteca.

Exemplo (em biblioteca com o contexto CAL):

```
TYPE Day : (  
MONDAY,  
TUESDAY,  
WEDNESDAY,  
THIRSDAY,  
FRIDAY,  
SATURDAY,  
SUNDAY) ;  
Declaração:  
eToday: CAL.Day;
```

Uso na aplicação:

```
IF eToday = CAL.Day.MONDAY THEN
```

### *Funções, Blocos Funcionais, Programas (POU) e Ações*

Os nomes das funções, blocos funcionais e programas consistem no prefixo da biblioteca (exemplo: COM), um sublinhado e um breve nome (exemplo: SENDTELEGRAM) da POU. Assim como com as variáveis, sempre a primeira letra do nome da POU é maiúscula e as demais, minúsculas. Recomenda-se que o nome da POU seja composto de um verbo e um substantivo.

Exemplo:

```
FUNCTION_BLOCK Com_SendTelegram (* Prefixo: comst *)
```

Na parte de declaração, deve ser fornecida uma breve descrição da POU como um comentário. Posteriormente, em todas as entradas e saídas também deve haver comentários. No caso de blocos funcionais, o prefixo associado para configurar a instância deve vir diretamente após o nome.

As ações não têm prefixo. Somente as ações que devem ser chamadas apenas internamente, ou seja, pela POU em si, iniciam com prv\_.

### *POUs nas Bibliotecas do MasterTool IEC XE*

Os nomes de POU nas bibliotecas do MasterTool IEC XE não precisam de um prefixo de biblioteca, pois os contextos fazem as vezes deste. Para criar nomes de método, as mesmas regras aplicam-se para ações. As possíveis entradas de um método devem estar comentadas para sua documentação. Também uma breve descrição do método deve ser adicionada a esta declaração.

### Nomes de Visualização

Deve-se evitar nomear uma visualização com o mesmo nome de outro objeto no projeto, pois isto pode causar problemas no caso de alterações na visualização.

### Inicialização de Variáveis

O valor de inicialização padrão é 0 para todas as declarações, porém valores de inicialização definidos pelo usuário podem ser adicionados na declaração de cada variável e tipo de dado.

A inicialização definida pelo usuário é produzida pelo operador de atribuição “:=” e pode ser qualquer expressão ST válida. Desta forma, valores constantes, assim como outras variáveis ou funções podem ser usadas para definir o valor de inicialização. O programador deve assegurar que a variável usada para a inicialização de outra variável já tenha sido inicializada.

Exemplos para inicializações de variáveis válidas:

```
VAR
  var1:INT := 12; (* Variável inteira com valor inicial 12*)

  x : INT := 13 + 8; (* Valor inicial definido por uma expressão com constantes *)

  y : INT := x + fun(4); (* Valor inicial definido por uma expressão contendo uma chamada de função. Neste caso, atente para a ordem *)

  z : POINTER TO INT := ADR(y); (* Não descrito pela norma IEC 61131-3: valores iniciais definidos por uma função de endereço. Neste caso, o ponteiro não será inicializado durante uma alteração online! *)

END_VAR
```

Consulte os itens: **ARRAYS, Estruturas, Tipos Subrange e Expressões Arbitrárias para Inicialização de Variáveis.**

NOTA: As variáveis das listas de variáveis globais são sempre inicializadas antes das variáveis locais de uma POU.

### Expressões Arbitrárias para Inicialização de Variáveis

Uma variável pode ser inicializada com qualquer expressão ST válida. É possível acessar outras variáveis externas ao escopo, assim como também é possível chamar funções. No entanto, o programador deve certificar-se de que a variável usada para inicialização de outra variável já foi inicializada.

Exemplos para inicializações de variáveis válidas:

```
VAR
  x : INT := 13 + 8;
  y : INT := x + fun(4);
  z : POINTER TO INT := ADR(y); (* Cuidado: o ponteiro não será inicializado na alteração online! *)
END_VAR
```

### Editor de Declaração

O editor de declaração é um editor textual ou tabular usado para a declaração de variáveis. Usualmente é fornecido em combinação com os editores de linguagem.

Consulte o capítulo sobre o **Editor de Declaração** no Manual de Utilização MasterTool IEC XE – MU299048 para obter mais informações.

### Diálogo Autodeclarar

Pode ser definido no diálogo *Opções*, categoria *Editor Textual*, que o diálogo *Autodeclarar* seja aberto assim que uma string ainda não declarada for inserida na parte de implementação de um editor e a tecla <ENTER> for pressionada. Este diálogo auxiliará na declaração da variável.

Este diálogo também pode ser explicitamente aberto pelo comando *Autodeclarar*, disponível no menu *Editar*, ou ainda pressionando as teclas <SHIFT>+<F2>. Se uma variável declarada já tiver sido selecionada anteriormente, sua declaração pode ser editada no diálogo.

### Modo Atalho

O editor de declaração, assim como outros editores textuais onde são feitas as declarações, permitem o modo de atalho.

Este modo é ativado quando a linha de uma declaração é finalizada com <CTRL>+<ENTER> e permite o uso de atalhos, ao invés de digitar a declaração completa, por extenso.

Os seguintes atalhos são suportados:

- Todos identificadores de uma linha, incluindo o último, se tornarão identificadores de variáveis declaradas.
- O tipo de declaração é determinado pelo último identificador da linha. Neste contexto, aplica-se o seguinte:

Identificador	Resultado
B ou BOOL	resulta em BOOL
I ou INT	resulta em INT
R ou REAL	resulta em REAL
S ou string	resulta em STRING

**Tabela 4-2. Tipo de Declaração**

- Se nenhum tipo tiver sido estabelecido segundo estas regras, automaticamente o tipo definido será BOOL e não o último identificador.
- Toda constante, dependendo do tipo da declaração, se tornará uma inicialização ou uma string.
- Um endereço (como em %MD12) recebe o atributo AT.
- Um texto após um ponto e vírgula (;) torna-se um comentário.
- Todos outros caracteres na linha são ignorados.

Exemplos:

Atalho	Declaração
A	A: BOOL;
A B I 2	A, B: INT := 2;
ST S 2; string A	ST:STRING(2); (* String A *)
X %MD12 R 5; Número real	X AT %MD12: REAL := 5.0;(* Número real *)
B !	B: BOOL;

**Tabela 4-3. Exemplos de Atalhos**

### Declaração AT

Para vincular uma variável de projeto diretamente a um endereço definido, este pode ser inserido na declaração AT da variável. A atribuição de uma variável a um endereço também pode ser feita no diálogo de mapeamento de um dispositivo na configuração do CP (árvore de dispositivos).



Sintaxe:

```
<Identificador> AT <Endereço> : <Tipo de dado>;
```

A palavra-chave AT deve ser seguida por um endereço válido. Veja o item **Endereço** para obter mais informações e possíveis sobreposições em caso de modo de endereçamento de bytes.

Esta declaração permite atribuir um nome específico a um endereço. Quaisquer alterações referentes aos sinais de entrada e saída podem ser feitas apenas em uma única posição (por exemplo, na declaração).

O seguinte deve ser observado ao escolher uma variável para ser atribuída a um endereço:

- Variáveis que requerem uma entrada não podem ser acessadas por escrita. O compilador caracteriza esse acesso como um erro.
- Declarações AT somente podem ser usadas com variáveis locais ou globais (e não com variáveis de entrada e saída de POU's).
- Declarações AT não devem ser usadas juntamente com VAR RETAIN ou VAR PERSISTENT, a não ser que exista algum mecanismo no CP utilizado que permita este tipo de operação.
- Se declarações AT forem usadas com estruturas ou membros de blocos funcionais, todas as instâncias acessarão a mesma posição de memória, o que corresponde às variáveis estáticas nas linguagens de programação clássicas (linguagem C, por exemplo).
- O layout de memória de estruturas também é determinado pelo dispositivo.

Exemplos:

```
counter_heat7 AT %QX0.0: BOOL;  
lightcabinetimpulse AT %IX7.2: BOOL;  
download AT %MX2.2: BOOL;
```

NOTA: Se variáveis booleanas são atribuídas a um Byte ou endereços WORD / DWORD, elas ocupam um byte com TRUE ou FALSE e não somente o primeiro bit após o offset.
--

### Palavras-chave

Palavras-chave são escritas em letras maiúsculas em todos os editores.

As seguintes strings são consideradas palavras-chave, isto é, não podem ser usadas como identificadores para variáveis ou POU's:

ABS, ACOS, ACTION (somente no formato Exportar), ADD, ADR, AND, ARRAY, ASIN, AT, ATAN, BITADR, BOOL, BY, BYTE, CAL, CALC, CALCN, CASE, CONSTANT, COS, DATE, DINT, DIV, DO, DT, DWORD, ELSE, ELSIF, END\_ACTION (somente no formato Exportar), END\_CASE, END\_FOR, END\_FUNCTION (somente no formato Exportar), END\_FUNCTION\_BLOCK (somente no formato Exportar), END\_IF, END\_PROGRAM (somente no formato Exportar), END\_REPEAT, END\_STRUCT, END\_TYPE, END\_VAR, END\_WHILE, EQ, EXIT, EXP, EXPT, FALSE, FOR, FUNCTION, FUNCTION\_BLOCK, GE, GT, IF, INDEXOF, INT, JMP, JMPC, JMPCN, LD, LDN, LE, LINT, LN, LOG, LREAL, LT, LTIME, LWORD, MAX, MIN, MOD, MOVE, MUL, MUX, NE, NOT, OF, OR, PARAMS, PERSISTENT, POINTER, PROGRAM, R, REFERENCE, READ\_ONLY, READ\_WRITE, REAL, REPEAT, RET, RETAIN, RETC, RETCN, RETURN, ROL, ROR, S, SEL, SHL, SHR, SIN, SINT, SIZEOF, SUPER, SQRT, ST, STN, STRING, STRUCT, SUPER, SUB, TAN, THEN, TIME, TO, TOD, TRUE, TRUNC, TYPE, UDINT, UINT, ULINT, UNTIL, USINT, VAR, VAR\_ACCESS, (usadas eventualmente, dependendo do hardware), VAR\_CONFIG, VAR\_EXTERNAL, VAR\_GLOBAL, VAR\_IN\_OUT, VAR\_INPUT, VAR\_OUTPUT, VAR\_STAT, VAR\_TEMP, WHILE, WORD, WSTRING e XOR.

Adicionalmente, todos os operadores de conversão listados no *Assistente de Entrada* são tratados como palavras-chave.

**Variáveis Locais VAR**

Entre as palavras-chave VAR e END\_VAR, devem ser declaradas todas as variáveis locais de uma POU. Estas variáveis não têm conexão externa, ou seja, não podem ser escritas externamente.

Considere a possibilidade de adicionar um atributo à VAR.

Exemplo:

```
VAR
iLoc1:INT; (* 1. Variável local*)
END_VAR
```

**Variáveis de Entrada - VAR\_INPUT**

Entre as palavras-chave VAR\_INPUT e END\_VAR, devem ser declaradas todas as variáveis que servem como variáveis de entrada para uma POU. Isto significa que, na posição da chamada, o valor das variáveis pode ser especificado na mesma.

Considere a possibilidade de adicionar um atributo à VAR\_INPUT.

Exemplo:

```
VAR_INPUT
iIn1:INT (* 1. Variável de entrada *)
END_VAR
```

**Variáveis de Saída - VAR\_OUTPUT**

Entre as palavras-chave VAR\_OUTPUT e END\_VAR, todas as variáveis que servem como variáveis de saída de uma POU devem ser declaradas. Isto significa que estes valores são retornados pela POU chamada. Ali, elas podem ser respondidas e usadas posteriormente.

Considere a possibilidade de adicionar um atributo à VAR\_OUTPUT.

Exemplo:

```
VAR_OUTPUT
iOut1:INT; (* 1. Variável de saída *)
END_VAR
```

**Variáveis de Saída em Funções e Métodos**

De acordo com a IEC 61131-3 versão 2, funções (e métodos) podem ter saídas adicionais. Estas devem ser atribuídas na chamada da função da seguinte forma:

```
fun(iIn1 := 1, iIn2 := 2, iOut1 => iLoc1, iOut2 => iLoc2);
```

O valor de retorno da função fun será escrito para as variáveis localmente declaradas loc1 e loc2, por exemplo.

**Variáveis de Entrada e Saída - VAR\_IN\_OUT**

Entre as palavras-chave VAR\_IN\_OUT e END\_VAR, devem ser declaradas todas as variáveis que servem como variáveis de entrada e saída para uma POU.

NOTA: Com variáveis deste tipo, o valor da variável transferida é alterado (transferido como um ponteiro, chamada-por-referência). Isto significa que o valor de entrada para estas variáveis não pode ser uma constante. Por esta razão, mesmo as variáveis VAR\_IN\_OUT de um bloco funcional não podem ser lidas nem escritas diretamente de fora via <instância do bloco funcional><variável de entrada/saída>.

Exemplo:

```
VAR_IN_OUT  
iInOut1:INT; (* 1. Variável de entrada e saída *)  
END_VAR
```

### Variáveis Globais - VAR\_GLOBAL

Variáveis normais, constantes, externas ou variáveis remanentes são conhecidas através do projeto e podem ser declaradas como variáveis globais.

#### NOTAS:

Uma variável definida localmente em uma POU com o mesmo nome de uma variável global tem prioridade na POU. As variáveis globais sempre serão inicializadas antes das variáveis locais das POU.

As variáveis devem ser declaradas localmente entre as palavras-chave VAR\_GLOBAL e END\_VAR. Considere a possibilidade de adicionar um atributo à VAR\_GLOBAL .

Uma variável é reconhecida como global pelo ponto precedente, por exemplo: .iGlobVar1.

Para informações detalhadas sobre o múltiplo uso dos nomes das variáveis, operador de escopo global “.” e contextos, consulte **Operador de Escopo Global**.

Use a lista de variáveis globais (GVLs) para estruturar e tratar variáveis globais em um projeto. Uma GVL pode ser adicionada via comando *Acrescentar Objeto*.

### Variáveis Temporárias - VAR\_TEMP

Este item é uma extensão da norma IEC 61131-3.

Declarações VAR\_TEMP somente são possíveis em programas e blocos funcionais. Estas variáveis também são acessáveis em um corpo de programa ou bloco funcional. Variáveis declaradas VAR\_TEMP são (re)inicializadas toda vez que a POU é chamada.

As variáveis devem ser declaradas localmente entre as palavras-chave VAR\_TEMP e END\_VAR.

### Variáveis Estáticas - VAR-STAT

Este item é uma extensão da norma IEC 61131-3.

Pode-se usar variáveis estáticas em blocos funcionais, métodos e funções. Elas devem ser declaradas localmente entre as palavras-chave VAR\_STAT e END\_VAR e serão inicializadas na primeira chamada da respectiva POU.

Variáveis estáticas somente são acessadas no escopo no qual são declaradas (como uma variável estática em C), porém, como são globais, não perdem seu valor após a POU ser finalizar a execução de uma chamada. Em uma função, por exemplo, elas podem ser usadas como contador para o número de chamadas da mesma.

Considere a possibilidade de adicionar um atributo à VAR\_STAT.

### Variáveis Externas – VAR\_EXTERNAL

São variáveis globais importadas para uma POU.

Elas devem ser declaradas localmente entre as palavras-chave VAR\_EXTERNAL e END\_VAR e na lista de variáveis globais. A declaração deve corresponder exatamente à declaração global. Se a variável global não existir, uma mensagem de erro será exibida.

NOTA: No MasterTool IEC XE, não é necessário definir as variáveis como externas. Esta palavra-chave é fornecida para manter a compatibilidade com a norma IEC 61131-3.

Exemplo:

```
VAR_EXTERNAL
iVarExt1:INT; (* Primeira variável externa *)
END_VAR
```

### Atributo Palavras-chave para Tipos de Variáveis

As palavras-chave de atributo listadas abaixo podem ser adicionadas à declaração do tipo das variáveis para especificar o escopo:

- RETAIN: Veja **Variáveis Remanentes** do tipo RETAIN
- PERSISTENT: Veja **Variáveis Remanentes** do tipo PERSISTENT
- CONSTANT: Veja **Constantes**

### Variáveis Remanentes

Variáveis remanentes podem reter os seus valores durante o período de execução normal do programa.

A declaração determina o grau de resistência de uma variável remanente no caso de reset, envio ou reinicialização do CP. Principalmente nas aplicações, será requisitado uma combinação de ambas as memórias remanentes.

Veja os seguintes itens:

- **Variáveis Retentivas**
- **Variáveis Persistentes**

#### Variáveis Retentivas

Variáveis declaradas como retentivas serão mantidas dependentes do CP, mas em uma área separada de memória. Elas apresentam a palavra-chave RETAIN na sua declaração em uma POU e em uma lista de variáveis globais.

Exemplo:

```
VAR RETAIN
iRem1 : INT; (* 1. Variável retentiva*)
END_VAR
```

Variáveis retentivas são identificadas pela palavra-chave RETAIN. Estas variáveis mantêm seu valor após o desligamento do controlador (inesperado ou normal) e no comando online *Reset a Quente*. Quando o programa for executado novamente, os valores armazenados serão processados posteriormente.

Todas outras variáveis são novamente inicializadas, tanto com seus valores inicializados, quanto com suas inicializações padrão.

Ao contrário das variáveis persistentes, as retentivas são reinicializadas em um novo envio do programa.

Variáveis retentivas, entretanto, são reinicializadas em um *Reset Origem* e, diferentemente das persistentes, em um *Reset a Frio* e em um envio da aplicação.

A propriedade *Retentiva* pode ser combinada com a propriedade *Persistente*.

#### NOTAS:

- Se uma variável local em um programa for declarada como VAR RETAIN, ela será salva na área retentiva (como um variável retentiva global).
- Se uma variável local em um bloco funcional for declarada como VAR RETAIN, a instância completa do bloco funcional será salva na área retentiva (todos os dados da POU), onde somente a variável retentiva declarada será tratada como retentiva.
- Se uma variável local em uma função for declarada como VAR RETAIN, então isto não terá efeito. A variável não será salva na área retentiva. Se uma variável local for declarada como persistente em uma função, então a mesma não terá efeito também.

### Variáveis Persistentes

Variáveis persistentes são identificadas pela palavra-chave PERSISTENT (VAR\_GLOBAL PERSISTENT). Elas são reinicializadas somente após um comando de *Reset Origem*. Ao contrário das variáveis retentivas, elas mantêm o seu valor após um envio.

Esse tipo de variáveis SOMENTE podem ser declaradas em uma lista de variáveis globais especial do tipo de objeto *Variáveis Persistentes*, atribuída a uma aplicação. Somente pode haver uma única lista.

NOTA: Uma declaração com VAR\_GLOBAL PERSISTENT tem o mesmo efeito que uma declaração com VAR\_GLOBAL PERSISTENT RETAIN ou VAR\_GLOBAL RETAIN PERSISTENT.

Assim como as variáveis retentivas, as persistentes são armazenadas em uma área de memória separada.

Exemplo:

```
VAR GLOBAL PERSISTENT RETAIN
iVarPers1 : DINT; (* 1. Variável Persistente+Retentiva Appl *)
bVarPers  : BOOL; (* 2. Variável Persistente+Retentiva Appl *)
END_VAR
```

NOTA: Somente é possível o uso de variáveis persistentes globais, a não ser que exista algum mecanismo no CP utilizado que permita este tipo de operação.

O dispositivo deve fornecer uma área separada de memória para a lista de variáveis persistentes de cada aplicação.

A cada reenvio da aplicação, a lista de variáveis persistentes no CP será verificada no projeto. A lista no CP é identificada pelo nome da aplicação. Se houver inconsistências, o usuário deverá reinicializar todas as variáveis persistentes da aplicação. Inconsistências podem ser resultantes de renomeação ou exclusão de outras modificações na lista de declarações existentes.

NOTA: Todas as modificações na parte de declaração da lista de variáveis persistentes, assim como o efeito da reinicialização resultante, devem ser cuidadosamente consideradas.

Novas declarações somente podem ser adicionadas ao final da lista. Em um envio, elas serão detectadas como novas e não demandarão uma reinicialização da lista completa.

Legenda do comportamento: valor é mantido (x) e valor é inicializado (-).

Situação	Comportamento		
Após comando online	VAR	VAR RETAIN	VAR PERSISTENT VAR RETAIN PERSISTENT VAR PERSISTENT RETAIN
Reset a Quente	-	x	x
Reset a Frio	-	-	x
Reset Origem	-	-	-
Envio da Aplicação	-	-	x
Alteração Online da Aplicação	x	x	x
Reinicializar CP	-	x	x

Tabela 4-4. Comportamento de Variáveis Remanentes

## Constantes

Constantes são identificadas pela palavra-chave **CONSTANT** e podem ser declaradas local ou globalmente.

Sintaxe:

```
VAR CONSTANT
<Identificador>:<Tipo> := <Inicialização>;
END_VAR
```

Exemplo:

```
VAR CONSTANT
c_iCon1:INT:=12; (* 1. Constante*)
END_VAR
```

Consulte a descrição dos **Operandos** para obter uma lista das possíveis constantes. Veja também a possibilidade de usar constantes tipadas.

## Literais Tipados

Para constantes IEC, será usado o menor tipo de dado possível. Outros tipos de dados podem ser obtidos com a ajuda de literais tipados, sem a necessidade de declarar explicitamente as constantes. Para isto, a constante será fornecida com um prefixo que determina o tipo.

Sintaxe:

```
<Tipo>#<Literal>
<Tipo> especifica o tipo de dado desejado (somente o tipo simples é possível). O tipo deve ser escrito em letras maiúsculas.
<Literal> especifica a constante. O dado inserido deve adequar-se ao tipo de dado especificado em <Tipo>.
```


Exemplo:

```
iVar1:=DINT#34;
```

Se a constante não pode ser convertida no tipo desejado sem perda de dados, a seguinte mensagem de erro é exibida:

Literais tipados podem ser usados da mesma forma que as constantes.

### Constantes no Modo Online

As constantes serão indicadas pelo símbolo  antes do valor, na coluna correspondente da janela de *Declaração* ou *Monitoração* no modo online. Neste caso, elas não podem ser acessadas por forçamento ou escrita, por exemplo.

### Configuração de Variáveis – VAR\_CONFIG

A configuração de variáveis pode ser usada para mapear as variáveis dos blocos funcionais na imagem do processo, isto é, nas E/S do dispositivo, sem a necessidade de especificar o endereço definido na declaração da variável do bloco funcional. O endereço, neste caso, é atribuído de forma centralizada para todas as instâncias do bloco funcional na lista global VAR\_CONFIG.

Para realizar esta configuração, atribua endereços incompletos às variáveis do bloco funcional declaradas entre as palavras-chave VAR e END\_VAR. Estes endereços são especificados com um asterisco.

Sintaxe:

```
<Identificador> AT %<I|Q>* : <Tipo de dado>;
```

Exemplo de uso de endereços não completamente definidos:

```
FUNCTION_BLOCK locio
VAR
xLocIn AT %I*: BOOL := TRUE;
xLocOut AT %Q*: BOOL;
END_VAR
```

Neste exemplo, são definidas duas variáveis locais de E/S: In (%I\*) local e Out (%Q\*) local.

Assim, a definição final do endereço deve ser feita na lista de variáveis globais, na Configuração de variáveis.

Para isto, adicione o objeto *Lista de Variáveis Globais (GVL)* à janela de dispositivos, através do comando *Acréscimo de Objeto*. Em seguida, insira as declarações das variáveis da instância com os endereços definidos entre as palavras-chave VAR\_CONFIG e END\_VAR.

As variáveis da instância devem ser especificadas com o caminho da instância completo, onde as POU's individuais e os nomes das instâncias são separados uns dos outros por pontos. A declaração deve conter um endereço cuja classe (entrada/saída) corresponda àquela do endereço não completamente definido, especificado (%I\*, %Q\*) no bloco funcional. O tipo de dado também deve concordar com a declaração no bloco funcional.

Sintaxe:

```
<Caminho da variável da instância> AT %<I|Q><Local> : <Tipo de dado>;
```

Variáveis de configuração cujo caminho da instância seja inválido pelo fato de a mesma não existir são registradas como erros. Por outro lado, um erro também será reportado se não existir configuração de endereço definida para uma variável da instância atribuída a um endereço incompleto.

Exemplo de configuração de variável:

Suponha que a seguinte definição para o bloco funcional “locio” seja determinada em um programa (veja no exemplo acima).

```
PROGRAM MainPrg
VAR
locioVar1: locio;
locioVar2: locio;
END_VAR
```

A configuração da variável corrigida teria a seguinte aparência:

```
VAR_CONFIG
MainPrg.locioVar1.xLocIn AT %IX1.0 : BOOL;
MainPrg.locioVar1.xLocOut AT %QX0.0 : BOOL;
MainPrg.locioVar2.xLocIn AT %IX1.0 : BOOL;
MainPrg.locioVar2.xLocOut AT %QX0.3 : BOOL;
END_VAR
```

NOTA: Alterações nas E/S diretamente mapeadas são imediatamente exibidas na imagem do processo, onde as alterações nas variáveis mapeadas via VAR\_CONFIG não são exibidas antes do final da tarefa considerada.

### Declaração e Inicialização de Tipos de Dados Definidos pelo Usuário

Além dos tipos de dados padrão, podem ser usados tipos definidos pelo usuário.

Para obter mais informações sobre a declaração e inicialização, consulte **Tipos de Dados** e **Tipos de Dados Definidos pelo Usuário**.

### Métodos FB\_Init e FB\_Reinit

#### FB\_Init

O método FB\_Init substitui o operador INI. Trata-se de um método especial para blocos funcionais que pode ser declarado explicitamente, mas que também está disponível implicitamente. Assim, em qualquer caso, ele pode ser acessado pelo bloco funcional.

Este método contém o código de inicialização do bloco funcional conforme declarado na parte de declaração deste. Se este método for declarado explicitamente, o código de inicialização implícito será inserido neste método. O programador pode adicionar outros códigos de inicialização.

NOTA: Quando a execução alcançar o código de inicialização definido pelo usuário, o bloco funcional estará completamente inicializado via código de inicialização implícita.

O método Init é chamado após o envio para cada instância declarada.

**ATENÇÃO:**  
Na alteração online, os valores anteriores substituirão os valores de inicialização.

Na sequência de chamada, em caso de herança, consulte: **FB\_Exit**.

Considere também a possibilidade de definir um método de instância de bloco funcional a ser chamado automaticamente após a inicialização via FB\_init: atributo call\_after\_init.

### Interface do Método Iniciar

```
METHOD fb_init : BOOL
VAR_INPUT
bInitRetains : BOOL; // Se TRUE, as variáveis retentivas serão
inicializadas (a quente / a frio).
bInCopyCode : BOOL; // Se TRUE, a instância será movida para o código
copiado posteriormente (alteração online).
END_VAR
```

O valor de retorno não é interpretado.

NOTA: Considere o uso de um método de saída e a ordem de execução resultante. Veja: **FB\_Exit**.



### Entrada Definida pelo Usuário

No método Init, entradas adicionais podem ser declaradas. Elas devem estar atribuídas a uma instância de um bloco funcional.

Exemplo de uso do método Init para o bloco funcional “serialdevice”:

```
METHOD fb_init : BOOL
VAR_INPUT
bInitRetains : BOOL; (* Inicialização das retentivas*)
bInCopyCode : BOOL; (* Instância movida para o código copiado *)
nCOMnum : INT; (* Entrada adicional: número da interface COM a ser
ouvida*)
END_VAR
```

Exemplo de declaração do bloco funcional “serialdevice”:

```
COM1 : serialdevice(nCOMnum:=1);
COM0 : serialdevice(nCOMnum:=0);
```

### *FB\_Reinit*

Se um método chamado FB\_reinit for declarado em uma instância de um bloco funcional, este método será chamado quando a instância for copiada (que é o caso, por exemplo, de uma alteração online após a modificação da declaração do bloco funcional). O método causará uma reinicialização do novo módulo da instância criado pelo código copiado. Os dados da instância original são escritos para a nova instância após a cópia e, se o usuário desejar os valores iniciais originais, pode valer-se da reinicialização. Este método não tem entradas. Considere que, ao contrário do FB\_init, este método deve ser declarado explicitamente. Se uma reinicialização da implementação do bloco funcional básico for desejada, FB\_reinit deve ser explicitamente chamado para aquela POU.

O método FB\_reinit não tem entradas.

Na sequência de chamada, em caso de herança, consulte: **FB\_Exit**.

### **FB\_Exit**

FB\_Exit é um método especial para blocos funcionais. Ele deve ser declarado explicitamente (não há declaração implícita). O método de saída - se estiver presente - é chamado para todas as instâncias declaradas dos blocos funcionais antes de um novo envio, em um reset ou durante uma alteração online para todas as instâncias movidas ou excluídas.

Há somente um parâmetro mandatório para o método de saída.

```
METHOD fb_exit : BOOL
VAR_INPUT
```

```
bInCopyCode : BOOL; // se TRUE, o método de saída é chamado para uma instância copiada após a
alteração online.
```

```
END_VAR
```

Considere também o método FB\_init e a seguinte ordem de execução.

- Método de saída - saída da antiga instância: old\_inst.fb\_exit(bInCopyCode := TRUE)
- Método de inicialização - inicialização da nova instância: new\_inst.fb\_init(bInitRetains := FALSE, bInCopyCode := TRUE)
- Copiar valores de blocos funcionais (copiar código): copy\_fub(&old\_inst, &new\_inst)

Além disto, em caso de herança, a seguinte sequência de chamada é verdadeira (para as POUs usadas nesta listagem, por exemplo, supõe-se SubFB EXTENDS MainFB e SubSubFB EXTENDS SubFB):

```
fbSubSubFb.FB_Exit(...);  
fbSubFb.FB_Exit(...);  
fbMainFb.FB_Exit(...);  
fbMainFb.FB_Init(...);  
fbSubFb.FB_Init(...);  
fbSubSubFb.FB_Init(...);
```

Para FB\_reinit:

```
fbMainFb.FB_reinit(...);  
fbSubFb.FB_reinit(...);  
fbSubSubFb.FB_Init(...);
```

### Instruções de Pragma

Uma instrução de pragma é usada para simular as propriedades de uma ou mais variáveis referentes aos processos de compilação e pré-compilação. Isto significa que um pragma influencia a geração do código. Por exemplo, ele pode determinar se uma variável será inicializada, monitorada, adicionada à lista de parâmetros, tornada invisível no gerenciador de bibliotecas ou adicionada à configuração de símbolos. Saídas de mensagem durante o processo de compilação podem ser forçadas, assim como podem ser usados pragmas condicionais - definindo como a variável será tratada, dependendo de determinadas condições. Estes pragmas também podem ser inseridos como Defines nas propriedades de compilação de um objeto específico.

Um pragma pode ser usado em uma linha separada ou com um texto suplementar em uma linha de implementação ou declaração do editor. Nos editores FBD/LD/IL, use o comando *Inserir Rótulo* e substitua o texto. Para definir um rótulo e um pragma, insira primeiro este último e depois o rótulo.

A instrução do pragma fica entre chaves e o uso de letras maiúsculas/minúsculas é indiferente.

```
{ <Texto da instrução> }
```

A chave que abre vem imediatamente após o nome da variável. A abertura e o fechamento das chaves deve ser na mesma linha.

Dependendo do seu tipo e conteúdo, o pragma tanto opera na linha na qual está localizado quanto nas linhas subsequentes e se mantém em operação até que seja finalizado por um pragma apropriado, até que seja executado com diferentes parâmetros ou ainda até que atinja o final do arquivo. Um arquivo neste contexto é uma parte de declaração, implementação, lista de variáveis globais ou declaração de tipo.

Se o compilador não conseguir interpretar a instrução, o pragma inteiro será tratado como um comentário, ignorando-o. Uma mensagem de advertência será apresentada neste caso.

Veja os seguintes tipos de pragma:

- Pragmas de mensagem
- Pragmas de atributo
- Pragmas condicionais

#### *Pragma de Mensagem*

Pragmas de mensagem podem ser usados para forçar a saída de mensagens em uma janela de *Mensagens* durante a compilação do projeto.

A instrução de pragma pode ser inserida em uma linha existente ou em uma linha separada no editor textual de uma POU. Pragmas de mensagem posicionados nas seções atualmente não definidas do código de implementação não serão considerados quando o projeto for compilado. Sobre isto, veja o exemplo fornecido com a descrição do Pragma condicional “defined (Identificador)”.

Existem quatro tipos de pragmas de mensagens:

Pragma	Tipo de mensagem
{text 'string'}	Texto: a string especificada será exibida.
{info 'string'}	📘 Informação: a string especificada será exibida.
{warning digit 'string'}	Advertência 🟡: a string especificada será exibida. Diferentemente de um Pragma obsoleto tipo de dado global, esta advertência é explicitamente definida para a posição local.
{error 'string'}	Erro 🛑: a string especificada será exibida.

Tabela 4-5. Tipos de Pragas de Mensagem

NOTA: No caso de mensagens dos tipos Informação, Advertência e Erro, a posição da fonte da mensagem - que é onde o pragma está localizado em uma POU - pode ser alcançada através do comando *Próxima Mensagem*. Isto não é possível para o tipo *Text*.

Exemplo de declaração e implementação no Editor ST:

```
VAR
    ivar : INT; {info 'TODO: deve ter outro nome'}
    bvar : BOOL;
    arrTest : ARRAY [0..10] OF INT;
    i:INT;
END_VAR
arrTest[i] := arrTest[i]+1;
ivar:=ivar+1;
{text 'A parte xy foi compilada completamente.'}
{info 'Isto é uma informação.'}
{warning 'Isto é uma advertência.'}
{error 'Isto é um erro.'}
```

The screenshot shows a 'Messages' window with a 'Build' dropdown menu. The status bar indicates '1 error(s)', '1 warning(s)', and '2 message(s)'. The main table lists the following messages:

Description	Project	Object	Position
----- Build started: Application: Device.Application -----			
typify code ...			
📘 TODO: deve ter outro nome	Untitled83	MainPrg [Device: PLC Logic: Applica...	Line 3 (Decl)
A parte xy foi compilada completamente.	Untitled83	MainPrg [Device: PLC Logic: Applica...	Line 3, Column 1 (Impl)
📘 Isto é uma informação.	Untitled83	MainPrg [Device: PLC Logic: Applica...	Line 4, Column 1 (Impl)
🟡 Isto é uma advertência.	Untitled83	MainPrg [Device: PLC Logic: Applica...	Line 5, Column 1 (Impl)
🛑 Isto é um erro.	Untitled83	MainPrg [Device: PLC Logic: Applica...	Line 6, Column 1 (Impl)
Compile complete -- 1 errors, 1 warnings			

Figura 4-1. Saída na Janela de Mensagens

### Pragma de Atributo

Pragas de atributo podem ser atribuídos a uma assinatura para influenciar a compilação e a pré-compilação, isto é, a geração de código.

Existem pragmas definidos pelos usuários que podem ser usados em combinação com pragmas condicionais e também existem os pragmas de atributo pré-definidos como padrões.

- Atributo 'displaymode'
- Atributo 'global\_init\_slot'
- Atributo 'hide'
- Atributo 'hide\_all\_locals'
- Atributo 'Init\_Namespace'
- Atributo 'init\_on\_onlchange'
- Atributo 'instance-path'
- Atributo 'linkalways'
- Atributo 'Monitoring'

- Atributo 'no\_check'
- Atributo 'noinit'
- Atributo 'obsolete'
- Atributo 'pack\_mode'
- Atributo 'qualified\_only'
- Atributo 'reflection'
- Atributo 'symbol'

### Atributos Definidos pelo Usuário

Atributos de pragmas podem ser atribuídos a POU's, declarações de tipo ou variáveis e podem ser definidos por usuários arbitrários ou através de uma aplicação. Estes atributos podem ser selecionados antes da compilação por meio de pragmas condicionais.

Sintaxe:

```
{attribute 'atributo'}
```

Esta instrução de pragma é válida para a linha atual ou para a subsequente (esta última no caso do cursor estar posicionado em uma linha separada. Consulte o item **Pragmas Condicionais**).

Um atributo definido pelo usuário pode ser atribuído aos seguintes objetos:

- POU's e Ações
- Variáveis
- Tipos

Exemplo de POU's e ações:

Atributo 'displaymode' para a variável dwVar1:

```
VAR
{attribute 'displaymode':='hex'}
dwVar1: DWORD;
END_VAR
```

Exemplo para variáveis:

Atributo 'DoCount', definido em **Operadores de Compilação Condicional**, para a variável ivar é adicionado:

```
PROGRAM MAINPRG
VAR
{attribute 'DoCount'}
ivar:INT;
bvar:BOOL;
END_VAR
```

### Atributo Displaymode

Com a ajuda do pragma {attribute 'displaymode'}, pode ser definido o modo de exibição de uma variável única. Esta configuração substituirá a configuração global para o modo de exibição de todas as variáveis de monitoração, feita através dos comandos do submenu *Modo de Exibição* (disponível no menu *Comunicação*).

O pragma deve ser posicionado na linha acima daquela que contém a declaração das variáveis.

Sintaxe:

```
{attribute 'displaymode':='<modo de exibição>'}
```

Para exibição no formato binário, são possíveis as seguintes definições:

```
{attribute'displaymode':='bin'}
{attribute'displaymode':='binary'}
```

Para exibição no formato decimal:

```
{attribute'displaymode':='dec'}
{attribute'displaymode':='decimal'}
```

Para exibição no formato hexadecimal:

```
{attribute'displaymode':='hex'}
{attribute'displaymode':='hexadecimal'}
```

Exemplo:

```
VAR
{attribute 'displaymode':='hex'}
dwVar1: DWORD;
END_VAR
```

### Atributo Global\_init\_slot

O pragma {attribute 'global\_init\_slot'} é aplicado somente em assinaturas. Por padrão, a ordem de inicialização das variáveis de uma lista de variáveis globais é arbitrária. Entretanto, é necessário definir a ordem para o caso, por exemplo, das variáveis de uma lista global estarem referindo-se a variáveis de outra lista. Neste caso, usa-se o pragma para organizar a ordem de inicialização.

Sintaxe:

```
{attribute 'global_init_slot' := '<Valor>'}
```

O modelo <Valor> deve ser substituído por um valor inteiro descrevendo a ordem de inicialização da assinatura. O valor padrão é 50000. Um valor menor faz com que a inicialização ocorra antes. No caso de assinaturas com o mesmo valor para o atributo 'global\_init\_slot', a sequência de inicialização destas fica indefinida.

Exemplo:

Suponha que o projeto inclui duas listas de variáveis globais (GVL\_1 e GVL\_2).



**Figura 4-2. Listas de Variáveis Globais**

As variáveis B e C são membros da lista de variáveis globais GVL\_1, seus valores iniciais dependem da variável A.

```
VAR_GLOBAL
B : INT:=A+1;
C : INT:=A-1;
END_VAR
```

A variável global A é membro da lista de variáveis globais GVL\_2.

```
{attribute 'global_init_slot' := '300'}
VAR_GLOBAL
A : INT:=1000;
END_VAR
```

Configurando-se o atributo 'global\_init\_slot' da GVL\_2 para 300 (menor valor na ordem de inicialização), assegura-se que a expressão "A + 1" esteja definida quando da inicialização de B e C,

dessa forma pode-se usar essas variáveis normalmente em qualquer POU do projeto. Se o pragma for retirado da GVL\_2 e as variáveis da GVL\_1 forem utilizadas em alguma POU será exibido um erro na compilação do projeto.

### Atributo Hide

Com a ajuda do pragma {attribute 'hide'}, é possível evitar que variáveis ou até mesmo assinaturas inteiras sejam exibidas na funcionalidade *Listar Componentes*, no diálogo *Assistente de Entrada* ou na parte de declaração no modo online. Somente a variável subsequente ao pragma será oculta.

Sintaxe:

```
{attribute 'hide'}
```

Para ocultar todas as variáveis locais de uma assinatura, use este atributo.

Exemplo:

O bloco funcional myPOU é implementado com o atributo:

```
FUNCTION_BLOCK myPOU
VAR_INPUT
a:INT;
{attribute 'hide'}
a_invisible: BOOL;
a_visible: BOOL;
END_VAR
VAR_OUTPUT
b:INT;
END_VAR
VAR
END_VAR
```

No programa principal, são definidas duas instâncias deste bloco funcional:

```
PROGRAM mainprg
VAR
POU1, POU2: myPOU;
END_VAR
```

Ao atribuir-se um valor de entrada para POU1 (a função *Listar Componentes* em POU1, por exemplo), na parte de implementação de MainPrg, serão exibidas as variáveis A, A\_visible e B. A variável oculta A\_invisible não será exibida.

### Atributos Hide\_all\_locals

Com a ajuda do pragma {attribute 'hide\_all\_locals'} é possível evitar que todas as variáveis locais de uma assinatura sejam exibidas na funcionalidade *Listar Componentes*, no diálogo *Assistente de Entrada*. Este atributo tem utilização idêntica ao do atributo hide para cada uma das variáveis locais.

Sintaxe:

```
{attribute 'hide_all_locals'}
```

Exemplo:

O bloco funcional myPOU é implementado com este atributo.

```
{attribute 'hide_all_locals'}
FUNCTION_BLOCK myPOU
VAR_INPUT
a:INT;
END_VAR
VAR_OUTPUT
b:BOOL;
END_VAR
VAR
```

```
c, d: INT;  
END_VAR
```

No programa principal, são definidas duas instâncias deste bloco funcional:

```
PROGRAM MainPrg  
VAR  
POU1, POU2: myPOU;  
END_VAR
```

Ao atribuir-se um valor de entrada para POU1 (a função *Listar Componentes* em POU1, por exemplo), na parte de implementação de MainPrg, serão exibidas as variáveis A e B, mas não as variáveis locais ocultas C ou D.

### Atributo Init\_Namespace

Uma variável do tipo STRING ou WSTRING, fornecida com o pragma {attribute 'init\_namespace'}, será inicializada com o contexto atual, ou seja, com o caminho da instância do bloco funcional em que está contida esta variável. Aplicar este pragma pressupõe o uso do atributo adicional 'noinit' para a variável de string e o atributo 'reflection' para o bloco funcional correspondente.

Sintaxe:

```
{attribute 'init_namespace'}
```

Exemplo:

O bloco funcional (POU) é fornecido com todos os atributos necessários.

```
{attribute 'reflection'}  
FUNCTION_BLOCK POU  
VAR_OUTPUT  
{attribute 'no_init'}  
{attribute 'instance-path'}  
myStr: STRING;  
END_VAR
```

No programa principal MainPrg, está definida a instância fb do bloco funcional (POU).

```
PROGRAM MAINPRG  
VAR  
fb: POU;  
newString: STRING;  
END_VAR  
newString:=fb.myStr;
```

A variável myStr será inicializada com o contexto atual, por exemplo, Device.Application.MainPrg.fb. Este valor será atribuído à newString no programa principal.

### Atributo Init\_on\_onlchange

O pragma {attribute 'init\_on\_onlchange'}, quando anexado a uma variável faz com que esta seja inicializada a cada alteração online.

Sintaxe:

```
{attribute 'init_on_onlchange'}
```

### Atributo Instance-path

O pragma {attribute 'instance-path'} pode ser adicionado a uma variável do tipo string local que, conseqüentemente, será inicializada com o caminho da árvore de dispositivos da POU a qual esta variável de string pertence. Isto pode ser útil no caso de mensagens de erro. Aplicar este pragma pressupõe o uso do atributo 'reflection' para a POU correspondente e o atributo adicional 'noinit' para a variável de string.

Sintaxe:

```
{attribute 'instance-path'}
```

Exemplo:

Suponha que o bloco funcional abaixo contenha o atributo 'reflection':

```
{attribute 'reflection'}
FUNCTION_BLOCK POU
VAR
    {attribute 'instance-path'}
    {attribute 'noinit'}
    str: STRING;
END_VAR
```

Uma instância do bloco funcional é chamada no programa MainPrg:

```
PROGRAM MAINPRG
VAR
myPOU: POU;
myString: STRING;
END_VAR
myPOU();
myString:=myPOU.str;
```

A instância myPOU, tendo sido inicializada com a variável de string STR, será atribuída ao caminho da instância myPOU da POU (por exemplo, Device.Application.MainPrg.myPOU). Este caminho será atribuído à variável myString no programa principal.

NOTA: O tamanho de uma variável string pode ser definida arbitrariamente (até 255), entretanto, esta string será truncada (a partir de seu final) se for atribuída a uma variável de tipo de dados menor.

### Atributo Linkalways

O pragma {attribute 'linkalways'} permite selecionar as POUs no compilador de forma que elas sempre sejam incluídas nas informações de compilação. Como resultado, os objetos com esta opção sempre serão compilados e enviados para o CP. Esta opção refere-se somente a POUs e GVLs localizadas em uma aplicação ou a bibliotecas inseridas nesta. A opção do compilador *Vincular Sempre* faz o mesmo.

Sintaxe:

```
{attribute 'linkalways'}
```

As POUs marcadas são usadas como base para as variáveis selecionáveis para a configuração de símbolos ao usar o editor correspondente.

Exemplo:

A lista de variáveis globais GVLMoreSymbols é implementada com o uso deste atributo.

```
{attribute 'linkalways'}
VAR_GLOBAL
g_iVar1: INT;
g_iVar2: INT;
END_VAR
```

### Atributo Monitoring

Uma propriedade pode ser monitorada no modo online de um editor ou através da janela *Monitoração*.

A monitoração pode ser ativada através da adição do atributo de pragma de monitoração na linha acima da definição da propriedade. Desta forma, na visualização online da POU em uma lista de



monitoração, o nome, tipo e valor das variáveis de propriedade serão exibidos. Portanto, os valores preparados podem ser inseridos para forçar as variáveis pertencentes à propriedade.

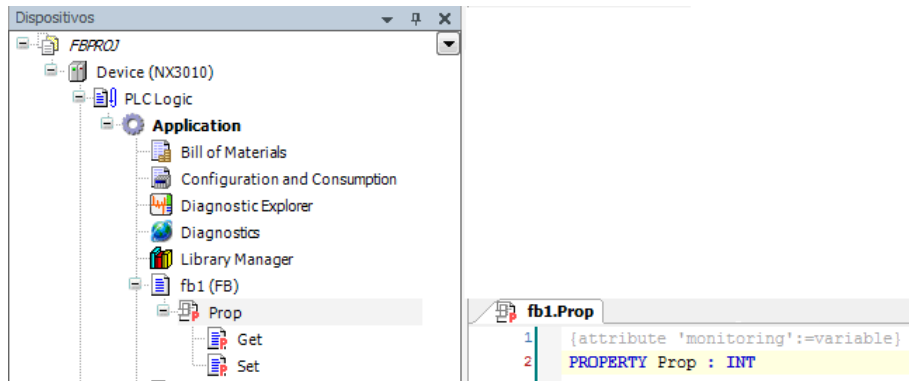


Figura 4-3. Exemplo de Propriedade Preparada para Variável de Monitoração

Expressão	Tipo	Valor	Valor Preparado	Comentário
SP1	REAL	60.42		
SP2	REAL	45.36		

1	SP2	45.4	:= 45.36;	
2	RETURN			

Figura 4-4. Exemplo de Visualização de Monitoração

Existem duas maneiras de monitorar o valor atual das variáveis de propriedade. Para o uso específico, considere qual atributo melhor se adequa para obter o valor desejado. Isto dependerá de quais operações nas variáveis são implementadas na propriedade.

Pragma { attribute 'monitoring':='variable' }

Pragma { attribute 'monitoring':='call' }

No primeiro caso ( { attribute 'monitoring':='variable' } ), uma variável implícita é criada para a propriedade, que obterá o valor da propriedade atual sempre que a aplicação chamar os métodos Set ou Get. O valor desta variável implícita será monitorado.

No segundo caso ( { attribute 'monitoring':='call' } ), o atributo pode ser usado para propriedades que retornam tipos de dados simples ou ponteiros (não para tipos estruturados).

O valor a ser monitorado é recuperado diretamente através da chamada da propriedade, ou seja, o serviço de monitoração do sistema de execução chama o método Get. Considere que, se quaisquer operações nas variáveis forem implementadas na propriedade, o valor ainda poderá ser alterado.

#### Atributo No\_check

O pragma { attribute 'no\_check' } é adicionado com a finalidade de suprimir a chamada de quaisquer **POUs para Verificações Implícitas**. As funções de verificação podem influenciar a performance e, por isto, é razoável aplicar este atributo no início de POU's já aprovadas ou que são chamadas com frequência.

Sintaxe:

```
{attribute 'no_check' }
```

Exemplo:

```
{attribute 'no_check'}
PROGRAM MainPrg
VAR
x: DINT (100..200);
y: DINT;
END_VAR
x := y;
```

Com a POU para verificação implícita `CheckRangeSigned`, por exemplo, adicionada ao projeto, execute o código abaixo com o pragma `{attribute 'no_check'}`, a função não será checada nesse caso, permitindo a variável “x” receber qualquer valor.

### Atributo `No_init`

Variáveis fornecidas com o pragma `{attribute 'no_init'}` não são inicializadas implicitamente. O pragma pertence à variável subsequente declarada.

Sintaxe:

```
{attribute 'no_init'}
```

ou

```
{attribute 'no-init'}
{attribute 'noinit'}
```

Exemplo:

```
PROGRAM mainprg
VAR
A : INT;
{attribute 'no_init'}
B : INT;
END_VAR
```

Se for executado um reset na aplicação associada, a variável inteira A será novamente inicializada implicitamente com 0 e a variável B manterá o valor ao qual está atualmente atribuído.

### Atributo `Obsolete`

Um pragma `{attribute 'obsolete'}` pode ser adicionado a uma definição de tipo de dado para originar uma advertência definida pelo usuário durante a execução da compilação, se o respectivo tipo de dado for usado no projeto (estrutura, bloco funcional, etc.). Assim, por exemplo, é possível anunciar que um determinado tipo de dado não deve mais ser usado.

Diferentemente de um pragma de mensagem, esta advertência é determinada na definição e, assim sendo, de forma global para todas as instâncias do tipo de dado.

Esta instrução de pragma sempre é válida para a linha atual ou - se localizado em uma linha separada - para a linha subsequente.

Sintaxe:

```
{attribute 'obsolete' := 'Texto definido pelo usuário'}
```

Exemplo:

O pragma obsoleto é inserido na definição do bloco funcional fb1.

```
{attribute 'obsolete' := 'Tipo de dado fb1 inválido'}
FUNCTION_BLOCK fb1
VAR_INPUT
i:INT;
END_VAR
...
```

Se `fb1` for usado como tipo de dado em uma declaração, por exemplo, `fbinst: fb1`, a seguinte advertência será exibida quando o projeto for compilado: Tipo de dado `fb1` inválido.

#### Atributo `Pack_mode`

O pragma `{attribute 'pack_mode'}` define o modo com que uma estrutura de dados é empacotada durante a alocação. O atributo deve ser configurado no topo de uma estrutura de dados e influenciará o empacotamento da estrutura inteira.

Sintaxe:

```
{attribute 'pack_mode':= '<Valor>'}
```

O modelo `<Valor>` incluído em cotas únicas deve ser substituído por um dos seguintes valores disponíveis:

Valor	Descrição
0	alinhado, isto é, não haverá intervalos de memória.
1	1-byte-alinhado (idêntico ao alinhado).
2	2-byte-alinhado, isto é, o tamanho máximo de um intervalo de memória é 1 byte.
4	4-byte-alinhado, isto é, o tamanho máximo de um intervalo de memória é 3 bytes.
8	8-byte-alinhado, isto é, o tamanho máximo de um intervalo de memória é 7 bytes.

**Tabela 4-6. Atributo `Pack_mode`**

Exemplo:

```
{attribute 'pack_mode':= '1'}
TYPE myStruct:
STRUCT
Enable: BOOL;
Counter: INT;
MaxSize: BOOL;
MaxSizeReached: BOOL;
END_STRUCT
END_TYPE
```

Uma variável do tipo de dado `myStruct` será instanciada alinhada: se o endereço do seu componente `Enable` for `0x0100`, por exemplo, então o componente `Counter` seguirá no endereço `0x0101`, `MaxSize`, no `0x0103` e no `MaxSizeReached` no `0x0104`. Com `'pack_mode'=2` `Counter` estará em `0x0102`, `MaxSize` em `0x0104` e `MaxSizeReached` em `0x0105`.

NOTA: O atributo pode também ser aplicado a POUs, porém é preciso ter cuidado com esta aplicação devido a possíveis ponteiros internos nas mesmas.

#### Atributo `Qualified_only`

Após atribuir o pragma `{attribute 'qualified_only'}` na parte superior de uma lista de variáveis globais, estas somente podem ser acessadas usando o nome da variável global, por exemplo, `gvl.g_var`. Isto é válido inclusive para as variáveis do tipo enumeração e pode ser útil para evitar incompatibilidade de nome nas variáveis locais.

Sintaxe:

```
{attribute 'qualified_only'}
```

Exemplo:

Suponha a seguinte GVL (lista de variáveis globais) com o atributo `'qualified_only'`:

```
{attribute 'qualified_only'}
VAR_GLOBAL
iVar: INT;
END_VAR
```

Em uma POU MainPrg, por exemplo, esta lista de variáveis locais deve ser chamada com o prefixo GVL:

```
GVL.iVar:=5;
```

A seguinte chamada incompleta da variável resultará em erros de compilação:

```
iVar:=5;
```

### Atributo Reflection

O pragma {attribute 'reflection'} é anexado às assinaturas. Por questões de desempenho, este é um atributo obrigatório para POUs que apresentam o atributo caminho de instância.

Sintaxe:

```
{attribute 'reflection'}
```

Exemplo:

Veja o exemplo do atributo caminho da instância (Instance path).

### Atributo Symbol

O pragma {attribute 'symbol'} define quais variáveis devem ser tratadas no *Symbol Configuration*, o que significa que elas serão exportadas como símbolos para uma lista, um arquivo XML (<Nome do Projeto>.Device.Application.xml) no diretório do projeto e também para um arquivo não visível para o usuário e disponível no dispositivo para acesso externo, por exemplo, por um servidor OPC. As variáveis fornecidas com este atributo serão enviadas para o CP mesmo que não tenham sido configuradas ou não estejam visíveis no editor de configuração de símbolos.

O *Symbol Configuration* deve estar disponível como um objeto na árvore de *Dispositivos*.

Sintaxe:

```
{attribute 'symbol' := 'none' | 'read' | 'write' | 'readwrite'}
```

O acesso só é permitido em símbolos vindo de programas ou listas de variáveis globais. Para acessar um símbolo, o seu nome deve ser completamente especificado.

A definição do pragma pode ser atribuída a variáveis específicas ou a todas as variáveis declaradas em um programa:

- Para ser válido para uma única variável, o pragma deve estar posicionado na linha anterior à declaração das variáveis.
- Para ser válido para todas as variáveis contidas na parte de declaração de um programa, o pragma deve ser posicionado na primeira linha do editor de declaração. De qualquer forma, também neste caso as configurações para as variáveis específicas devem ser modificadas através da adição explícita de um pragma.

O possível acesso a um símbolo é definido pelo parâmetro de pragma 'none', 'read', 'write' ou 'readwrite'. Se nenhum parâmetro estiver definido, o padrão 'readwrite' será válido.

Exemplo:

Com a configuração abaixo, as variáveis A e B serão exportadas com acesso de leitura e escrita e a variável D será exportada com acesso de leitura.

```
{attribute 'symbol' := 'readwrite'}  
PROGRAM MAINPRG  
VAR  
A : INT;  
B : INT;  
{attribute 'symbol' := 'none'}  
C : INT;  
{attribute 'symbol' := 'read'}  
D : INT;
```

END\_VAR

### Pragmas Condicionais

A linguagem ExST (ST estendido) suporta várias instruções de pragma condicionais que afetam a geração de código nos processos de compilação e pré-compilação.

O código de implementação a ser considerado para estes processos depende de vários fatores:

- se um determinado tipo de dado ou variável está declarado
- se um tipo ou variável apresenta um determinado atributo
- se uma variável apresenta um determinado tipo de dado
- se uma determinada POU ou tarefa está disponível e é parte da árvore de chamadas

NOTA: Uma POU ou uma GVL declarada na árvore de POUs não pode usar a instrução {define...} declarada em uma aplicação.

Pragma	Descrição
<b>{define identifier string}</b>	Durante o pré-processamento, todas as instâncias subsequentes do identificador serão substituídas pela sequência de tokens se a string destes não estiver vazia (o que é permitido e definido). O identificador permanece definido e no escopo até o final do objeto ou até que seja indefinido pela diretiva {undefine}. Veja abaixo o exemplo Compilação condicional.
<b>{undefine identifier}</b>	A definição do identificador de pré-processamento ({define}, veja acima) será removida e o identificador se tornará indefinido. Se o identificador especificado não estiver definido atualmente, este pragma será ignorado.
<b>{IF expr}</b> ... <b>{ELSIF expr}</b> ... <b>{ELSE}</b> ... <b>{END_IF}</b>	Estes são pragmas para compilação condicional. As expressões especificadas expr devem ser uma constante no período de compilação. Elas são avaliadas na ordem em que aparecem, até que uma das expressões seja avaliada em um valor diferente de zero. O texto associado à diretiva bem sucedida será pré-processado e compilado normalmente; os outros serão ignorados. A ordem das seções é determinada, porém as seções elsif e else são opcionais e as seções elsif podem aparecer arbitrariamente com frequência. Na constante expr, vários operadores de compilação condicional podem ser usados (veja abaixo).

**Tabela 4-7. Pragmas Condicionais**

### Operadores de Compilação Condicional

Na expressão constante “expr” de um pragma de compilação condicional ({if} ou {elsif}), podem ser usados vários operadores, os quais podem não estar definidos ou redefinidos via {undefine} ou {define}, respectivamente. Estas expressões, assim como a definição feita em {define}, podem também ser usadas em *Definições do Compilador* no diálogo *Propriedades* de um objeto.

Os seguintes operadores são suportados:

Sintaxe	Descrição
<b>Defined (Identificador)</b>	Quando aplicado a um identificador, seu valor será TRUE se este identificador for definido com uma instrução {define} e esta não for revertida através de {undefine}. Caso contrário, o valor será FALSE. Exemplo: Pré-requisito: há duas POUs. A variável pdef1 é definida pela instrução {define} na POU_1, mas não em POU_2. Ambas possuem o código abaixo: {IF defined (pdef1)} (* Este código é processado na POU_1*) {info 'pdef1 defined'} ivar := ivar + SINT#1;

	<pre>{ELSE} (* Este código é processado na POU_2*) {info 'pdef1 not defined'} ivar := ivar - SINT#1; {END_IF} No exemplo, somente a string de informação 'pdef1 defined' será exibida na janela de mensagem quando a POU_1 for chamada, pois pdef1 está atualmente definida. A mensagem de informação 'pdef1 not defined' será exibida caso pdef1 não seja definida, POU_2.</pre>
<b>Defined (variable:nome da variável)</b>	<p>Quando aplicado a uma variável, seu valor será TRUE se esta for declarada no escopo atual. Caso contrário, será FALSE.</p> <p>Exemplo:</p> <p>Pré-requisito: há duas POU, POU_1 e POU_2. A variável g_bTest é declarada em POU_2, mas não em POU_1. Ambas possuem o código abaixo:</p> <pre>{IF defined (variable:g_bTest)} (* O seguinte código é processado somente na POU_2*) g_bTest := x &gt; 300; {END_IF}</pre>
<b>Defined (type:identificador)</b>	<p>Quando aplicado a um identificador de tipo, seu valor será TRUE se o tipo com este nome específico foi declarado. Caso contrário, será FALSE.</p> <p>Exemplo: Em um primeiro momento declare o tipo de dado DUT no projeto, em um segundo momento não declare, veja a diferença na variável booleana "bDutDefined".</p> <pre>{IF defined (type:DUT)} (* A seguinte linha de código somente será processada se houver o tipo declarado *) bDutDefined := TRUE; {END_IF}</pre>
<b>Defined (pou:nome da POU)</b>	<p>Quando aplicado a um nome de POU, seu valor será TRUE se a POU ou uma ação com este nome específico foram declaradas. Caso contrário, será FALSE.</p> <p>Exemplo:</p> <p>Pré-requisito: Em um cenário a POU CheckBounds foi adicionado ao projeto e em outra situação não. Em uma POU existe o código abaixo:</p> <pre>{IF defined (pou:CheckBounds)} (* A seguinte linha de código somente será processada no cenário em que há a POU CheckBounds *) arrTest[CheckBounds(0,i,10)] := arrTest[CheckBounds(0,i,10)] + 1; {ELSE} (* A seguinte linha de código somente será processada no cenário em que não há a POU CheckBounds *) arrTest[i] := arrTest[i]+1; {END_IF}</pre>
<b>Hasattribute (pou: nome da POU, 'atributo')</b>	<p>Quando aplicado a uma POU, seu valor será TRUE se este determinado atributo for especificado na primeira linha da parte de declaração das POU.</p> <p>Exemplo:</p> <p>Pré-requisito: há dois blocos funcionais, fun1 e fun2. Em fun1, adicionalmente, há o atributo 'vision'.</p> <p>Definição de fun1:</p> <pre>{attribute 'vision'} FUNCTION fun1 : INT VAR_INPUT i : INT; END_VAR VAR END_VAR Definição de fun2: FUNCTION fun1 : INT VAR_INPUT i : INT; END_VAR</pre>

	<pre> VAR END_VAR Em uma POU existe o código abaixo: {IF hasattribute (pou: fun1, 'vision')} (* A seguinte linha de código será processada*) ergvar := fun1(ivar); {END_IF} {IF hasattribute (pou: fun2, 'vision')} (* A seguinte linha de código não será processada*) ergvar := fun2(ivar); {END_IF} </pre>
<p><b>Hasattribute (variable: nome da variável, 'atributo')</b></p>	<p>Quando aplicado a uma variável, seu valor será TRUE se este determinado atributo for especificado via instrução {attribute} na linha anterior à declaração da variável.</p> <p>Exemplo:</p> <p>Pré-requisito: há duas POU, POU_1 e POU_2. A variável "g_globaln" é usada em ambas POU, porém em POU_1 adicionalmente há o atributo 'DoCount'.</p> <p>Declaração de g_globalnt em POU_1 :</p> <pre> VAR_GLOBAL {attribute 'DoCount'} g_globalnt : INT; g_multiType : STRING; END_VAR </pre> <p>Declaração de g_globalnt em POU_2 :</p> <pre> VAR_GLOBAL g_globalnt : INT; g_multiType : STRING; END_VAR {IF hasattribute (variable: g_globalnt, 'DoCount')} (* A seguinte linha de código somente será processada em POU_1, pois ali a variável "g_globalnt" apresenta o atributo 'DoCount' *) g_globalnt := g_globalnt + 1; {END_IF} </pre>
<p><b>Hastype (variable: nome da variável, tipo)</b></p>	<p>Quando aplicado a uma variável, seu valor será TRUE se esta determinada variável apresentar o tipo especificado. Caso contrário, será FALSE.</p> <pre> ANY ANY_DERIVED ANY_ELEMENTARY ANY_MAGNITUDE ANY_BIT ANY_STRING ANY_DATE ANY_NUM ANY_REAL ANY_INT LREAL REAL LINT DINT INT SINT ULINT UDINT UINT USINT TIME LWORD DWORD WORD BYTE BOOL </pre>

	<p>STRING WSTRING DATE_AND_TIME DATE TIME_OF_DAY</p> <p>Exemplo: Pré-requisito: Em uma POU, com o código abaixo, em um primeiro momento a variável g_multitype é declarada como LREAL e em um segundo momento é declarada como STRING.</p> <pre>{IF (hastype (variable: g_multitype, LREAL))} (*A seguinte linha de código somente será processada quando a variável for declarada como LREAL *) g_multitype := (0.9 + g_multitype) * 1.1; {ELSIF (hastype (variable: g_multitype, STRING))} (*A seguinte linha de código somente será processada quando a variável for declarada como STRING *) g_multitype := 'this is a multi-talented'; {END_IF}</pre>
<b>Hasvalue (define-ident, string)</b>	<p>Se "define" (define-ident) estiver configurado e apresentar o valor especificado (string), então seu valor será TRUE. Caso contrário, será FALSE.</p> <p>Exemplo: Pré-requisito: a variável test é usada em uma POU. Em um primeiro momento ela tem o valor "1" em um segundo momento tem o valor "2".</p> <pre>{IF hasvalue(test,'1')} (* A seguinte linha de código somente será processada quando a variável "test" apresenta o valor "1" *) x := x + 1; {ELSIF hasvalue(test,'2')} (* A seguinte linha de código somente será processada quando a variável "test" apresenta o valor "2" *) x := x + 2; {END_IF}</pre>
<b>NOT operador</b>	<p>Inverte o valor do operador.</p> <p>Exemplo: Pré-requisito: Em um cenário a função CheckBounds foi adicionado ao projeto e em outra situação não, a POU MainPrg existe em ambos os casos. Em uma POU existe o código abaixo:</p> <pre>{IF defined (pou: MainPrg) AND NOT (defined (pou: CheckBounds))} (* A seguinte linha de código somente será executada no cenário em que não há a função CheckBounds. *) bAndNotTest := TRUE; {END_IF}</pre>
<b>Operador AND operador</b>	<p>Será TRUE se ambos operadores forem TRUE.</p> <p>Exemplo: Pré-requisito: Em um cenário a função CheckBounds foi adicionado ao projeto e em outra situação não, a POU MainPrg existe em ambos os casos. Em uma POU existe o código abaixo:</p> <pre>{IF defined (pou: MAINPRG) AND (defined (pou: CheckBounds))} (* A seguinte linha de código somente será processada na primeiro cenário, pois somente ali MainPrg e CheckBounds estão definidos*) bAndTest := TRUE; {END_IF}</pre>
<b>Operador OR operador</b>	<p>Será TRUE se um dos operadores for TRUE.</p> <p>Exemplo: Pré-requisito: Em um cenário a função CheckBounds foi adicionado ao projeto e em outra situação não, a POU MainPrg existe em ambos os casos. Em uma POU existe o código abaixo:</p> <pre>{IF defined (pou: MAINPRG) OR (defined (pou: CheckBounds))} (* A seguinte linha de código será processada nos dois cenários, pois ambos contêm no mínimo uma das POU's *) bOrTest := TRUE; {END_IF}</pre>



(Operador)	Aplica "(" no operador.
------------	-------------------------

Tabela 4-8. Sintaxe de Defined

## Funcionalidade Listar Componentes

A norma IEC 61131-3 dá suporte a entradas de texto. As funcionalidades de *Listar Componentes* (menu *Ferramentas, Opções, SmartCoding*) auxiliam na inserção de um identificador correto.

- Se for inserido um ponto (“.”) em qualquer lugar onde puder ser inserido um identificador global, aparecerá uma caixa de seleção listando todas as variáveis globais disponíveis. Escolha um destes elementos e pressione a tecla <ENTER> para inseri-lo após o ponto. O elemento também pode ser inserido através de um duplo clique na lista.
- Inserindo uma instância de um bloco funcional ou uma variável de estrutura seguida de um ponto, aparecerá uma caixa de seleção listando todas as variáveis de entrada e saída do bloco funcional correspondente ou os componentes da estrutura. Escolha o elemento desejado e insira-o pressionando a tecla <ENTER> ou com um duplo clique na lista. Exemplos:

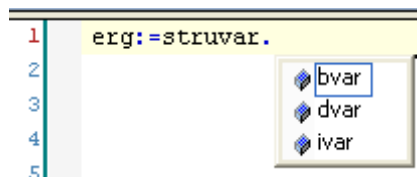


Figura 4-5. Listar Componentes (Componentes de uma Estrutura)

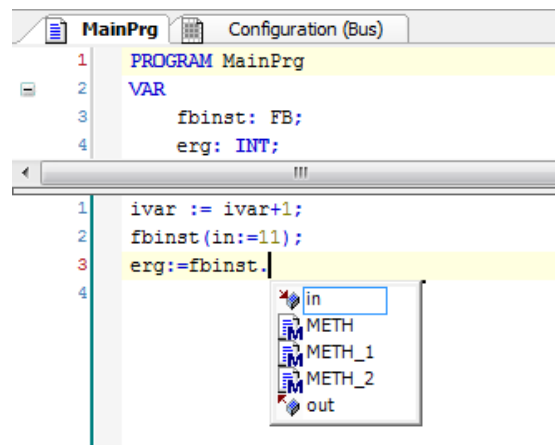


Figura 4-6. Listar Componentes (Componentes de um Bloco Funcional)

- Uma caixa de seleção listando todas as POU's e as variáveis globais disponíveis no projeto aparecerá quando for inserida uma string e as teclas <CTRL>+<SPACE> forem pressionadas. O primeiro item da lista, iniciando com a string dada, pode ser selecionado e inserido no programa através da tecla <ENTER>.

## Mapeamentos de E/S

Este sub-diálogo dos dispositivos é chamado *Bus: Mapeamento de E/S*, ele serve para configurar um mapeamento de E/S do CLP. Isto significa que as variáveis do projeto, utilizados pelas aplicações, são atribuídos aos endereços de entradas e saídas do dispositivo.

### Geral

Todo o mapeamento de E/S pode ser configurado para o dispositivo atual.

Um endereço também pode ser atribuído a uma variável através de declarações AT. Nesse caso considere os seguintes pontos:

Declarações AT podem ser usadas somente com as variáveis locais ou globais, no entanto, não com variáveis de entradas e saídas de POU's.

Para declarações AT não é possível gerar forçamentos para variáveis.

Se declarações AT são usados com estrutura ou bloco funcional, todas as instâncias terão acesso ao mesmo local de memória, o que corresponde a variáveis estáticas em linguagens de programação clássicas, como por exemplo C.

Para mais informações, consulte **Declaração AT**.

A tela *Bus: Mapeamento de E/S* apresenta em sua parte inferior os seguintes comandos:


- *Resetar Mapeamento*: Este botão reseta as configurações de mapeamento para os padrões definidos pelo arquivo de descrição de dispositivo.
- *Sempre atualizar variáveis*: Se esta opção for ativada, todas as variáveis serão atualizadas em cada ciclo da tarefa, não importa se eles são usados, ou se eles são mapeados para uma entrada ou saída.

## Canais

Variável	Mapeamento	Canal	Endereço	Tipo	Unidade	Descrição
		Reserved	%QB81921			Reserved for internal use.
		Reserved	%QB81921	BYTE		Reserved for internal use.
		Reserved	%QX81921.0	BOOL		
		Reserved	%QX81921.1	BOOL		
		Reserved	%QX81921.2	BOOL		
		Reserved	%QX81921.3	BOOL		
		Reserved	%QX81921.4	BOOL		
		Reserved	%QX81921.5	BOOL		
		Reserved	%QX81921.6	BOOL		
		Reserved	%QX81921.7	BOOL		
		Reserved	%QB81922	BYTE		Reserved for internal use.
		Reserved	%IW81924			Reserved for internal use.
		Reserved	%IW81924	WORD		Reserved for internal use.
		Reserved	%IW81926	WORD		Reserved for internal use.
		Reserved	%IW81928	WORD		Reserved for internal use.

**Figura 4-7. Janela de Canais**


- *Variável*: Pode ser mapeada uma variável para o canal, através do *Assistente de Entrada*, ou criada uma nova, através da edição do campo.
- *Mapeamento*: Indica se a variável será nova (🔗), será declarada internamente como uma variável global, ou será feito um mapeamento para uma já existente (🔗), neste caso o endereço será exibido riscado e não deve ser utilizado diretamente.
- *Canal*: Nome do canal de entrada ou saída do dispositivo.
- *Endereço*: Endereço do canal, por exemplo: “%IW0”.
- *Tipo*: Tipo de dado do canal de entrada ou saída, por exemplo “BOOL”.
- *Unidade*: Unidade do valor do parâmetro, por exemplo “MS” para milissegundos.
- *Descrição*: Texto de descrição para o parâmetro.
- *Valor Atual*: valor atual do parâmetro, exibido no modo online.


Você pode modificar e corrigir o campo *Endereço* de uma saída ou entrada. Para isso selecione a coluna de endereços e pressione a tecla <ESPAÇO> para abrir o campo de edição. Agora, modifique o valor ou deixe sem modificações e feche o campo de edição através da tecla <ENTER>. O campo de endereço ficará marcado pelo símbolo .

Canais						
Variável	Mapeamento	Canal	Endereço	Tipo	Unidade	Descrição
		Reserved	M %QB85000			Reserved for internal use.

**Figura 4-8. Exemplo de Endereço Manualmente Modificado**

Só é possível modificar o endereço de entrada ou de saída inteira, não os seus subelementos. Assim, se uma entrada ou saída está representado na tabela de mapeamento com qualquer subárvore, apenas o campo de endereço mais alto pode ser editado.

Se você quiser remover a fixação do valor, reabra a edição do campo *Endereço*, exclua o valor e feche com <ENTER>. O endereço e os endereços seguintes serão definido de volta para os valores que tinham antes da modificação manual, e o símbolo  será removido.

NOTA: Em projetos criados a partir do modelo *Projeto MasterTool Padrão*, os canais dos dispositivos serão modificados pelo MasterTool IEC XE com o objetivo de manter uma melhor distribuição e organização de canais. Portanto, eles aparecerão com o símbolo .

## Tipos de Dados

Para a programação, podem ser usados tipos de dados padrão, tipos de dados definidos pelo usuário ou instâncias de blocos funcionais. Cada identificador é atribuído a um tipo de dado, que determina quanto espaço de memória será reservado e quais tipos de valores ele armazena.

### Tipos de Dados Padrão

Todos os tipos de dados descritos na norma IEC 61131-3 são suportados pelo MasterTool IEC XE. Veja a seguir:

- BOOL / BIT
- Tipos de Dados Inteiros
- REAL / LREAL
- STRING / WSTRING
- Tipos de Dado de Tempo

Observe que, além dos tipos de dados extensivos à norma, também é possível definir tipos próprios (tipos de dados definidos pelo usuário).

### BOOL

As variáveis tipo BOOL podem apresentar os valores TRUE e FALSE. Serão reservados 8 bits de espaço de memória.

Veja também: **Constantes BOOL**.

### BIT

Assim como as variáveis do tipo BOOL, as variáveis tipo BIT podem apresentar valores TRUE e FALSE. Diferente do tipo BOOL, o tipo BIT ocupa apenas 1 bit de espaço de memória. Contudo para que essa alocação possa ocorrer de forma correta este tipo de dado só pode ser declarado em Blocos Funcionais ou Estruturas (definidas em objetos do tipo DUT).

### Tipos de Dados Inteiros

Veja abaixo uma lista de todos os tipos de dados inteiros disponíveis. Cada um dos diferentes tipos de dados cobre um diferente intervalo de valores. As seguintes limitações de intervalo aplicam-se aos tipos de dados inteiros:

Tipo	Limite inferior	Limite superior	Espaço de memória
BYTE	0	255	8 Bits

<b>WORD</b>	0	65535	16 Bits
<b>DWORD</b>	0	4294967295	32 Bits
<b>LWORD</b>	0	$2^{64}-1$	64 Bits
<b>SINT</b>	-128	127	8 Bits
<b>USINT</b>	0	255	8 Bits
<b>INT</b>	-32768	32767	16 Bits
<b>UINT</b>	0	65535	16 Bits
<b>DINT</b>	-2147483648	2147483647	32 Bits
<b>UDINT</b>	0	4294967295	32 Bits
<b>LINT</b>	$-2^{63}$	$2^{63}-1$	64 Bits
<b>ULINT</b>	0	$2^{63}-1$	64 Bits

Tabela 4-9. Tipos de Dados Inteiros

Como resultado, quando tipos maiores são convertidos em menores, informações podem ser perdidas.

Veja também: **Constantes Numéricas**.

### REAL/LREAL

REAL e LREAL são os chamados tipos de ponto flutuante. Eles são utilizados para representar números racionais. 32 bits de espaço de memória são reservados para REAL e 64 para LREAL.

Intervalo de valores para REAL: 1.401298e-45 a 3.402823466e+38.

Intervalo de valores para LREAL: 2.2250738585072014e-308 a 1.7976931348623157e+308.

#### NOTAS:

- O suporte ao tipo de dado LREAL depende do dispositivo. Consulte a documentação correspondente para verificar se o tipo LREAL de 64 bits persiste ou se será convertido para REAL durante a compilação (possivelmente com perda de informações).
- Se um tipo de dado REAL ou LREAL for convertido em SINT, USINT, INT, UINT, DINT, UDINT, LINT ou ULINT e o valor do número real estiver fora do intervalo de valor daquele inteiro, o resultado será indefinido e dependerá do dispositivo. Até uma exceção é possível neste caso. Para obter um código independente do dispositivo, gerencie qualquer excesso de limite da faixa pela aplicação. Se o número REAL/LREAL estiver na faixa de valor do inteiro, a conversão funcionará em todos os sistemas da mesma forma.

Veja também: **Constantes REAL/LREAL**.

### STRING

Uma variável do tipo STRING pode conter qualquer sequência de caracteres. O item tamanho, na declaração, determina quanto espaço de memória deve ser reservado para a variável. Trata-se do número de caracteres de uma string e pode ser posicionado entre parênteses ou colchetes. Se não for especificado tamanho, será usado o tamanho padrão de 80 caracteres.

O comprimento da string, basicamente, não é limitado no MasterTool IEC XE, mas as funções de string somente podem processar strings de 1 a 255 caracteres. Se uma variável for inicializada com uma string muito longa para os tipos de dados das variáveis, a string será truncada da direita para a esquerda.

Exemplo de uma declaração de string com 35 caracteres:

```
str:STRING(35):='Isto é uma String';
```

Veja também: **WSTRING** e **Constantes STRING**.

### *Tipos de Dado de Tempo*

Os tipos de dados TIME, TIME\_OF\_DAY (abreviação TOD), DATE e DATE\_AND\_TIME (abreviação DT) são tratados internamente como DWORD.

O tempo é dado em milissegundos em TIME e TOD, sendo que, neste último, o tempo inicia em 12 A.M.

O tempo é dado em segundos em DATE e DT iniciando em Janeiro 1, 1970 às 12:00 A.M.

Neste contexto, veja também neste manual:

- **LTIME** (disponível também como um tipo de dado de tempo de 32 bits)
- **Constantes TIME**
- **Constantes DATE**
- **Constantes DATE\_AND\_TIME**
- **Constantes TIME\_OF\_DAY**

### **Extensões da Norma IEC 1131-3**

#### *Tipos de Dados Adicionais à Norma*

Além dos tipos de dados em conformidade com a IEC 61131-3, há também alguns tipos extensivos a norma, disponíveis implicitamente no MasterTool IEC XE:

- UNION
- LTIME
- WSTRING
- Ponteiros
- Referências

#### **UNION**

Como uma extensão da norma IEC 61131-3, é possível declarar uniões em tipos definidos pelo usuário.

Em uma união, todos os componentes tem o mesmo offset, ou seja, todos eles ocupam a mesma posição de armazenagem. Assim, supondo a definição de união mostrada no exemplo a seguir, uma atribuição ao nome “.a” também afetaria o nome “.b”.

Exemplo:

```
TYPE name: UNION
a : LREAL;
b : LINT;
END_UNION
END_TYPE
```

#### **LTIME**

Como uma extensão à norma IEC 61131-3, LTIME é suportado como base de tempo para timers de alta resolução. O tamanho de LTIME é 64 bits e sua resolução é em nanosegundos.

Sintaxe:

```
LTIME#<Declaração de tempo>
```

A declaração de tempo pode incluir unidades de tempo, como as usadas com a constante TIME e, adicionalmente, microssegundos (us) e nanosegundos (ns), respectivamente.

Exemplo:

```
LTIME1 := LTIME#1000d15h23m12s34ms2us44ns
```

Compare com **Constantes TIME** (tamanho 32 bits e resolução em milissegundos).

## WSTRING

Este tipo de dado de string é uma extensão da norma IEC 61131-3.

Ele é diferente do tipo STRING padrão (ASCII) porque é interpretado no formato Unicode.

Exemplo:

```
wstr:WSTRING:="Isto é uma WString";
```

Veja também: **STRING** e **Constantes STRING** (Operandos).

## Ponteiros

Como uma extensão da norma IEC 61131-3, é possível usar ponteiros.

Ponteiros salvam os endereços das variáveis, programas, blocos funcionais, métodos e funções durante a execução de um programa aplicativo. Um ponteiro pode apontar para qualquer objeto e tipo de dado, inclusive os definidos pelo usuário. Observe a possibilidade de usar uma função de verificação de ponteiro implícita.

Sintaxe:

```
<Identificador>: POINTER TO <Tipo de dado | Bloco funcional | Programa |
Método | Função>;
```

Desreferenciar um ponteiro significa obter o valor atualmente armazenado no endereço para o qual ele está apontando. Um ponteiro pode ser desreferenciado adicionando-se o operador de conteúdo “^” após o identificador do ponteiro. Veja “pt^” no exemplo abaixo.

O operador de endereço ADR pode ser usado para atribuir o endereço de uma variável a um ponteiro.

Exemplo:

```
VAR
pt:POINTER TO INT; (* Declaração do ponteiro pt *)
var_int1:INT := 5; (* Declaração das variáveis var_int1 e var_int2 *)
var_int2:INT;
END_VAR
pt := ADR(var_int1); (* Endereço da varint1 é atribuído ao ponteiro pt *)
var_int2:= pt^; (* Valor 5 da var_int1 é atribuído a var_int2 via
desrefêrencia do ponteiro pt; *)
```

## Ponteiros de Função

Os ponteiros de função são suportados e substituem o operador INDEXOF. Estes ponteiros podem ser enviados para bibliotecas externas, porém não é possível chamar um ponteiro de função em uma aplicação no sistema de programação. A função de execução para registro das funções de chamada de retorno (função de biblioteca de sistema) aguarda o ponteiro de função, e, dependendo da chamada de retorno para a qual o registro foi requisitado, a respectiva função será chamada implicitamente pelo sistema de tempo de execução (por exemplo, em STOP). Para habilitar uma chamada de sistema (sistema de tempo de execução) a respectiva propriedade deve estar definida para o objeto da função.

O operador ADR pode ser usado em nomes de função, programa, blocos funcionais e métodos. Uma vez que funções podem mover-se após uma alteração online, o resultado não é o endereço da função, mas o endereço de um ponteiro para a função. Este endereço será válido enquanto a função existir no dispositivo.

Veja também: **INDEXOF**.

## Índices de Acesso para Ponteiros

Como extensão da norma IEC 61131-3, o índice de acesso “[ ]” é permitido às variáveis do tipo POINTER, STRING e WSTRING.

- Pint[i] retornará o tipo de dado de base.
- O índice de acesso para ponteiros é aritmético: se o índice de acesso for usado em uma variável do tipo ponteiro, o offset será calculado por:  $\text{pint}[i] = (\text{pint} + i * \text{SIZEOF}(\text{base type}))^\wedge$ . O índice

de acesso também executa uma desreferência no ponteiro. O tipo resultante é o tipo de base do ponteiro. Observe que `pint[7]` é diferente de `(pint + 7)^`.

- Se o índice de acesso for usado em uma variável do tipo `STRING`, o resultado é um caractere no offset `index-expr`. O resultado é do tipo `BYTE`. `Str[i]` retornará o “i-ésimo” caractere da string, assim como um `SINT` (ASCII).
- Se o índice de acesso for usado em uma variável do tipo `WSTRING`, o resultado é um caractere no offset `index-expr`. O resultado é do tipo `WORD`. `Wstr[i]` retornará o “i-ésimo” caractere da string, assim como um `INT` (Unicode).

### Função `CheckPointer`

Para verificar acessos a ponteiros durante a execução, use a função de verificação implícita disponível `CheckPointer`, chamada antes de cada acesso ao endereço de um ponteiro. Adicione o objeto *POUs para Verificações Implícitas* à aplicação usando o diálogo *Acréscimo de Objeto*. Marque a caixa de verificação referente ao tipo `CheckPointer` e confirme suas configurações com o botão *Abrir*, onde a função de verificação será aberta no editor. A parte de declaração é pré-definida e não pode ser modificada, exceto para adicionar outras variáveis locais. Entretanto, ao contrário de outras funções de verificação, não há implementação padrão de `CheckPointer` disponível, a implementação é deixada para o usuário.

A função `CheckPointer` deve verificar se o endereço ao qual o ponteiro refere-se está dentro do intervalo de memória válido. Além disto, o alinhamento da área de memória referenciada deve adequar-se ao tipo de dado da variável para a qual o ponteiro aponta. Se ambas as condições forem preenchidas, `CheckPointer` deve retornar o ponteiro de entrada inalterado. O tratamento adequado dos casos de erro é de responsabilidade do usuário.

Modelo:

Parte de declaração:

```
// Código implicitamente gerado: NÃO EDITAR.  
FUNCTION CheckPointer : POINTER TO BYTE  
VAR_INPUT  
ptToTest : POINTER TO BYTE;  
iSize : DINT;  
iGran : DINT;  
bWrite: BOOL;  
END_VAR
```

Parte de Implementação:

```
// Não há um modo padrão de implementação. Preencha o seu próprio código aqui.  
CheckPointer := ptToTest;
```

Ao ser chamada, a função apresenta os seguintes parâmetros de entrada:

- `ptToTest`: endereço de destino do ponteiro.
- `iSize`: tamanho da variável referenciada. O tipo de dado de `iSize` deve ser inteiro e compatível e deve cobrir o tamanho de dados potenciais máximo armazenados no endereço do ponteiro.
- `iGran`: granularidade do acesso, ou seja, o maior tipo de dado não estruturado usado na variável referenciada. O tipo de dado de `iGran` deve ser inteiro e compatível.
- `bWrite`: tipo do acesso (`TRUE`= acesso de escrita, `FALSE`= acesso de leitura). O tipo de dado de `bWrite` deve ser `BOOL`.
- Valor de retorno: endereço usado para desreferenciar o ponteiro (passado no primeiro parâmetro de entrada).

### Tipos de Dados Definidos pelo Usuário

Além dos tipos de dados padronizados, o usuário pode definir tipos de dados especiais em um projeto.

Estas definições são possíveis através da criação de objetos DUT (Unidades de tipos de dados) na janela de *POUs* e na parte de declaração de uma *POU*.

Observe as recomendações sobre a nomeação de um objeto para torná-lo o mais exclusivo possível.

Veja, a seguir, os tipos de dados definidos pelo usuário:

- ARRAYS
- Estruturas
- Enumerações
- Tipos Subrange
- Ponteiros

### ARRAYS

Campos (ARRAYS) uni, bi e tridimensionais são suportados como tipos de dados elementares. Os ARRAYS podem ser definidos tanto na parte de declaração de uma *POU* quanto nas listas de variáveis globais. Observe a possibilidade de usar verificações de limites.

Sintaxe:

```
<Nome>:ARRAY [<l11>..l1>,<l12>..l2>,<l13>..l3>] OF <Tipo>
```

l1, l2 e l3 identificam o limite inferior do intervalo.

u1, u2 e u3 identificam o limite superior.

Os valores do intervalo devem ser inteiros.

Exemplo:

```
Card_game: ARRAY [1..13, 1..4] OF INT;
```

### Inicialização de ARRAYS

#### ATENÇÃO:

Os colchetes devem ser incluídos na parte de inicialização.

Exemplo para a inicialização completa de um ARRAY:

```
arr1 : ARRAY [1..5] OF INT := [1,2,3,4,5];  
arr2 : ARRAY [1..2,3..4] OF INT := [1,3(7)]; (* Abreviação para 1,7,7,7 *)  
arr3 : ARRAY [1..2,2..3,3..4] OF INT := [2(0),4(4),2,3];  
(* Abreviação para 0,0,4,4,4,4,2,3 *)
```

Exemplo de inicialização de um ARRAY de uma estrutura:

Definição da estrutura:

```
TYPE STRUCT1  
STRUCT  
p1:int;  
p2:int;  
p3:dword;  
END_STRUCT  
END_TYPE
```

Inicialização do ARRAY:

```
ARRAY[1..3] OF STRUCT1:= [(p1:=1,p2:=10,p3:=4723),(p1:=2,p2:=0,p3:=299),  
(p1:=14,p2:=5,p3:=112)];
```

Exemplo de inicialização parcial de um ARRAY:

```
arr1 : ARRAY [1..10] OF INT := [1,2];
```



Elementos para os quais nenhum valor é pré-atribuído são inicializados com o valor inicial padrão do tipo básico. No exemplo acima, os elementos `arr1[3]` a `arr1[10]` são, portanto, inicializados com 0.

### Acessando Componentes de ARRAY

Componentes de ARRAY são acessados em uma ARRAY bidimensional usando-se a seguinte sintaxe:

```
<Nome>[Índice 1, Índice 2]
```

Exemplo:

```
Card_game [9,2]
```

### Check Functions

Para acessar elementos de ARRAY durante a execução, a função *CheckBounds* deve estar disponível durante a aplicação. Adicione o objeto *POUs para Verificações Implícitas* à aplicação através do diálogo *Acréscimo de Objeto*. Selecione a caixa de verificação referente a esta função e confirme as configurações com o botão *Abrir*. A função *CheckBounds* então será aberta no editor. A parte de declaração é pré-definida e não pode ser modificada (exceto para a adição de outras variáveis locais). No editor ST, é fornecida uma implementação padrão da função passível de modificação.

Esta função de verificação trata da violação de limites através de um método apropriado (por exemplo, configurando uma memória de erro detectada ou alterando o índice). A função será chamada implicitamente assim que a variável do tipo ARRAY for atribuída.

#### ATENÇÃO:

Para manter a funcionalidade de verificação, não altere a parte de declaração de uma função de verificação implícita.

Exemplo de uso da função *CheckBounds*:

Abaixo, encontra-se a implementação padrão da função de verificação.

Parte de declaração:

```
// Código implicitamente gerado: NÃO EDITAR.  
FUNCTION CheckBounds : DINT  
VAR_INPUT  
index, lower, upper: DINT;  
END_VAR
```

Parte de implementação:

```
// Código implicitamente gerado: somente uma sugestão de implementação.  
IF index < lower THEN  
CheckBounds := lower;  
ELSIF index > upper THEN  
CheckBounds := upper;  
ELSE  
CheckBounds := index;  
END_IF
```

Ao ser chamada, a função apresenta os seguintes parâmetros de entrada:

- Index: índice do elemento do campo
- Lower: limite inferior do intervalo
- Upper: limite superior do intervalo

Enquanto o índice estiver dentro do intervalo, o valor de retorno será o índice em si.

Caso contrário (violação do intervalo), serão retornados os limites inferior e superior.

No programa, o limite superior do ARRAY A é excedido:

```
PROGRAM MAINPRG
VAR
a: ARRAY[0..7] OF BOOL;
b: INT:=10;
END_VAR
a[b]:=TRUE;
```

Neste caso, a chamada implícita da função *CheckBounds* antes da atribuição faz com que o valor do índice seja alterado de 10 (dez) para o limite superior 7 (sete). Portanto, o valor TRUE será atribuído ao elemento A[7] do ARRAY. Esta é uma tentativa de, via função *CheckBounds*, corrigir a tentativa de acesso fora do intervalo.

### Estruturas

Estruturas são criadas como objetos *DUT* (Unidade de Tipo de Dado) via diálogo *Acrescentar Objeto*.

Elas iniciam com as palavras-chave *TYPE* e *STRUCT* e terminam com *END\_STRUCT* e *END\_TYPE*.

#### ATENÇÃO:

A palavra-chave *TYPE*, nas declarações de estrutura, deve ser seguida por um “:”.

A sintaxe para declarações de estrutura se dá como segue:

```
TYPE <Nome da estrutura>:
STRUCT
    <Declaração de variáveis 1>
    ...
    <Declaração de variáveis n>
END_STRUCT
END_TYPE
```

<Nome da estrutura> é um tipo reconhecido através do projeto e pode ser usado como um tipo de dado padrão.

Estruturas de intertravamento são permitidas. A única restrição é que as variáveis não podem ser atribuídas aos endereços (a declaração *AT* não é permitida).

Exemplo para uma definição de estrutura denominada *Polygone*:

```
TYPE Polygone:
STRUCT
Start:ARRAY [1..2] OF INT;
Point1:ARRAY [1..2] OF INT;
Point2:ARRAY [1..2] OF INT;
Point3:ARRAY [1..2] OF INT;
Point4:ARRAY [1..2] OF INT;
End:ARRAY [1..2] OF INT;
END_STRUCT
END_TYPE
```

### Inicialização de Estruturas

Exemplo de inicialização de uma estrutura:

```
Poly_1:polygone := ( Start:=[3,3], Point1:=[5,2], Point2:=[7,3],
Point3:=[8,5], Point4:=[5,7], End := [3,5]);
```

Inicializações com variáveis não são possíveis. Veja um exemplo da inicialização de um *ARRAY* de uma estrutura em *ARRAYS*.

## Acesso aos Componentes de Estrutura

Os componentes de estrutura podem ser acessados através da seguinte sintaxe:

```
<Nome da estrutura>.<Nome do componente>
```

Assim, para o exemplo acima mencionado da estrutura Polygonline, o componente Start pode ser acessado por Poly\_1.Start.

## Enumerações

Uma enumeração é um tipo de dado definido pelo usuário formado por um número determinado de constantes string. Estas constantes referem-se à valores de enumeração.

Valores de enumeração serão reconhecidos globalmente em todas as áreas do projeto, mesmo se estiverem declarados em uma POU.

Uma enumeração é criada como um objeto *DUT* através do diálogo *Acréscitar Objeto*.

### ATENÇÃO:

Não é possível a declaração de enumeração local, exceto com TYPE.

### Sintaxe:

```
TYPE <Identificador>:(<Enum_0> ,<Enum_1>, ..., <Enum_n>) |<Tipo de dado de base>;
END_TYPE
```

Uma variável do tipo <Identificador> pode assumir um dos valores de enumeração <Enum\_..> e ser inicializada com o primeiro. Estes valores são compatíveis com todos os números, o que significa que podem ser executadas operações com os mesmos, da mesma forma que com as variáveis inteiras. Um número "x" pode ser atribuído à variável. Se os valores de enumeração não forem inicializados com valores específicos na declaração, a contagem iniciará com 0 (zero). Ao inicializar, assegure-se de que os valores iniciais estão aumentando nos componentes da linha. A validade do número será verificada no momento da execução.

### Exemplo:

#### Definição de duas enumerações:

```
TYPE TRAFFIC_SIGNAL: (red, yellow, green:=10); (* O valor inicial para
cada cor é vermelho 0, amarelo 1 e verde 10 *)
END_TYPE
```

#### Declaração:

```
TRAFFIC_SIGNAL1 : TRAFFIC_SIGNAL;
```

#### Uso do valor de enumeração TRAFFIC\_SIGNAL em uma POU:

```
TRAFFIC_SIGNAL1:=0; (* O valor do sinal de trânsito é vermelho*)
```

## Extensões à Norma IEC 61131-3

O tipo enumeração pode ser usado (como um operador de escopo) para tornar não ambíguo o acesso a uma constante de enumeração.

Assim, é possível usar a mesma constante em diferentes enumerações.

### Exemplo:

#### Definição de duas enumerações:

```
TYPE COLORS_1: (red, blue);
END_TYPE
TYPE COLORS_2: (green, blue, yellow);
END_TYPE
```

Uso do valor de enumeração BLUE em uma POU:

Declaração:

```
colorvar1 : COLORS_1;
colorvar2 : COLORS_2;
```

Implementação:

```
(* É permitido: *)
colorvar1 := colors_1.blue;
colorvar2 := colors_2.blue;
(* Não é permitido: *)
colorvar1 := blue;
colorvar2 := blue;
```

O tipo de dado de base da enumeração (por padrão, INT), pode ser especificado.

Exemplo:

O tipo de dado de base para a enumeração BigEnum deve ser DINT:

```
TYPE BigEnum : (yellow, blue, green:=16#8000) DINT;
END_TYPE
```

### Tipos Subrange

Um tipo subrange é um tipo definido pelo usuário, cujo intervalo de valores é restrito em relação ao tipo de dado básico. Observe a possibilidade de usar verificações de limites de intervalo implícitas.

A declaração pode ser feita em um objeto DUT, mas também uma variável pode ser diretamente declarada com um tipo subrange.

Sintaxe para a declaração como um objeto DUT:

```
TYPE <Nome> : <Inttype> (<ug>..<og>) END_TYPE;
```

Sintaxe	Descrição
<Nome>	Deve ser um identificador IEC válido.
<Inttype>	É um dos tipos de dados SINT, USINT, INT, UINT, DINT, UDINT, BYTE, WORD, DWORD (LINT, ULINT, LWORD).
<ug>	É uma constante que deve ser compatível com o tipo de dado básico e que define o limite inferior dos tipos de intervalo. O limite inferior em si está incluído neste intervalo.
<og>	É uma constante que deve ser compatível com o tipo de dado básico e que define o limite superior dos tipos de intervalo. O limite superior em si está incluído neste intervalo.

**Tabela 4-10. Tipos Subrange**

Exemplos:

```
TYPE
SubInt : INT (-4095..4095);
END_TYPE
```

Declaração direta de uma variável com um tipo subrange:

```
VAR
i : INT (-4095..4095);
ui : UINT (0..10000);
END_VAR
```

Se, a um tipo subrange, estiver atribuído um valor (na declaração ou implementação) incompatível com este intervalo (no exemplo de declaração acima, i=5000), uma mensagem de erro será exibida.

## Funções de Verificação

Para verificar os limites de intervalo durante o tempo de execução, as funções `CheckRangeSigned` ou `CheckRangeUnsigned` devem estar disponíveis para a aplicação. Para isto, adicione o objeto *POUs para Verificações Implícitas* à aplicação via diálogo *Acréscentar Objeto*. Marque as caixas de verificação correspondentes aos tipos *CheckRangeSigned* e/ou *CheckRangeUnsigned* e confirme com *Abrir*. A função selecionada será aberta no editor. A parte de declaração é pré-definida e não pode ser modificada, exceto para adicionar outras variáveis locais. No editor ST, é fornecida uma implementação padrão da função passível de modificação.

O propósito desta função de verificação é o tratamento adequado das violações do subrange (por exemplo, ao configurar uma memória de erro detectada ou alterar o valor). A função será chamada implicitamente assim que uma variável do tipo subrange for atribuída.

**ATENÇÃO:**

Para manter a funcionalidade de verificação, não altere a parte de declaração de uma função de verificação implícita.

**Exemplo:**

A atribuição de uma variável pertencente a um tipo subrange assinado implica em uma chamada implícita à `CheckRangeSigned`. Na implementação padrão da função, é fornecido um valor para o intervalo conforme segue.

**Parte de declaração:**

```
// Código implicitamente gerado: NÃO EDITAR.
FUNCTION CheckRangeSigned : DINT
VAR_INPUT
value, lower, upper: DINT;
END_VAR
```

**Parte de implementação:**

```
// Código implicitamente gerado: somente uma sugestão de implementação.
IF (value < lower) THEN
CheckRangeSigned := lower;
ELSIF(value > upper) THEN
CheckRangeSigned := upper;
ELSE
CheckRangeSigned := value;
END_IF
```

Ao ser chamada, a função apresenta os seguintes parâmetros de entrada:

- Value: o valor a ser atribuído ao tipo de intervalo
- Lower: o limite inferior do intervalo
- Upper: o limite superior do intervalo

Enquanto o valor atribuído estiver no intervalo, a saída da função é o valor em si. Caso contrário, correspondentemente à violação do intervalo, os limites inferior e superior serão retornados.

A atribuição `i:=10*y` será substituída implicitamente por:

```
i := CheckRangeSigned(10*y, -4095, 4095);
```

Se `y` apresentar o valor 1000, por exemplo, `10000:= 10*1000` não será atribuída a variável `i`, conforme fornecido na implementação original, mas ao limite superior do intervalo, ou seja, 4095.

O mesmo se aplica à função `CheckRangeUnsigned`.

**NOTA:** Se as funções `CheckRangeSigned` ou `CheckRangeUnsigned` não estiverem presentes, não ocorrerá verificação de tipo dos tipos subrange durante a execução. Nesta caso, a variável `i` poderia obter qualquer valor entre -32768 e 32767.

### ATENÇÃO:

O uso das funções `CheckRangeSigned` e `CheckRangeUnsigned` podem resultar em um laço infinito, por exemplo, se um tipo subrange for usado como incremento de um laço FOR que não seja compatível com o intervalo.

Exemplo de um laço infinito:

```
VAR
ui : UINT (0..10000);
END_VAR
FOR ui:=0 TO 10000 DO
...
END_FOR
```

O laço FOR nunca será deixado, pois a função de verificação evitará que sejam atribuídos valores maiores que 10000 à variável “ui”.

## Operadores

### Operadores IEC e Funções Adicionais à Norma

O MasterTool IEC XE suporta todos operadores IEC. Diferentemente das funções padrão, estes operadores são reconhecidos implicitamente no projeto.

Além dos operadores IEC, também os seguintes operadores não prescritos pela norma são suportados: `ANDN`, `ORN`, `XORN`, `INDEXOF`, `SIZEOF` (consulte **Operadores Aritméticos**), `ADR`, `BITADR`, Operador de Conteúdo (consulte **Operadores de Endereço**) e alguns Operadores de Escopo.

Operadores são usados como funções em uma POU.

NOTA: Em operações com variáveis de ponto flutuante, o resultado depende do dispositivo em uso no momento.

Veja as seguintes categorias de operadores:

- Operadores de atribuição: `:=`, `MOVE`
- Operadores aritméticos
- Operadores de bitstring
- Operadores de deslocamento de bits
- Operadores de seleção
- Operadores de comparação
- Operadores de endereço
- Operadores de chamada
- Funções de conversão de tipo
- Funções numéricas
- Operadores adicionais à norma IEC
- Operadores de escopo adicionais à norma IEC

### Operadores Aritméticos

Os seguintes operadores, prescritos pela norma IEC61131-3, estão disponíveis:

- `ADD`
- `MUL`
- `SUB`
- `DIV`

- MOD
- MOVE

Além destes, há também dois outros operadores extensivos à norma:

- SIZEOF
- INDEXOF

## ADD

Operador IEC: adição de variáveis.

Tipos permitidos: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL e LREAL.

Duas variáveis TIME podem também ser somadas resultando em outra, por exemplo,  $t\#45s + t\#50s = t\#1m35s$ .

Exemplo em IL:

<b>LD</b>		7	
<b>ADD</b>		2	
<b>ADD</b>		4	
<b>ADD</b>		7	
<b>ST</b>		iVar	

Exemplo em ST:

```
var1 := 7+2+4+7;
```

Exemplo em FBD:



## MUL

Operador IEC: multiplicação de variáveis.

Tipos permitidos: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL e LREAL.

Exemplo em IL:

<b>LD</b>		7	
<b>MUL</b>		2	r
		4	r
		7	
<b>ST</b>		Var1	

Exemplo em ST:

```
var1 := 7*2*4*7;
```

Exemplo em FBD:



## SUB

Operador IEC: subtração de uma variável a partir de outra.

Tipos permitidos: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL e LREAL.

Uma variável TIME pode ser subtraída de outra variável TIME, resultando em um terceiro tipo de variável TIME. Considere que valores TIME negativos são indefinidos.

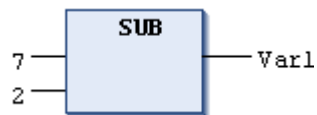
Exemplo em IL:

<b>LD</b>		7	
<b>SUB</b>		2	
<b>ST</b>		Var1	

Exemplo em ST:

```
var1 := 7-2;
```

Exemplo em FBD:



## DIV

Operador IEC: divisão de uma variável por outra.

Tipos permitidos: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL e LREAL.

Exemplo em IL:

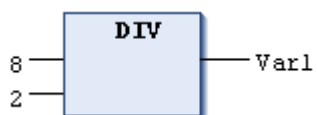
<b>LD</b>		8	
<b>DIV</b>		2	
<b>ST</b>		Var1	

O resultado em Var1 é 4.

Exemplo em ST:

```
var1 := 8/2;
```

Exemplo em FBD:



NOTA: Observe que diferentes dispositivos podem apresentar diferentes comportamentos em relação à divisão por zero.



NOTA: Observe que a utilização do operador DIV com tipos de dados inteiros retorna apenas o quociente da divisão. Caso se queira retornar o resto da divisão o operador a ser utilizado é o MOD descrito a seguir.

## Funções Check

Para verificar o valor do divisor, por exemplo, para evitar uma divisão por 0, é possível usar as funções de verificação CheckDivInt, CheckDivLint, CheckDivReal e CheckDivLReal. Após as mesmas terem sido incluídas na aplicação, cada divisão que ocorrer no código relacionado provocará uma chamada destas funções. Para incluí-las na aplicação, use o diálogo *Acréscitar Objeto*. A seguir, escolha o objeto *POUs para Verificações Implícitas*, selecione a caixa de seleção correspondente e confirme com *Abrir*. A função selecionada será aberta no editor. A parte de declaração das funções está pré-definida e não deve ser alterada (exceto para adicionar variáveis locais). Uma implementação padrão das funções modificáveis está disponível em ST.

Exemplo de implementação padrão da função CheckDivReal:

Parte de declaração:

```
// Código implicitamente gerado : NÃO EDITAR.
FUNCTION CheckDivReal : REAL
VAR_INPUT
divisor:REAL;
END_VAR
```

Parte de implementação:

```
// Código implicitamente gerado: somente uma sugestão para implementação.
IF divisor = 0 THEN
CheckDivReal:=1;
ELSE
CheckDivReal:=divisor;
END_IF;
```

O operador DIV usa a saída da função CheckDivReal como divisor. No exemplo seguinte, uma divisão por 0 é proibida. Assim, o valor 0 do divisor D é alterado para "1" (pela CheckDivReal) antes da execução da divisão. Portanto, o resultado da divisão é 799.

```
PROGRAM MainPrg
VAR
    erg:REAL;
    v1:REAL:=799;
    d:REAL;
END_VAR
erg:= v1 / d;
```

## MOD

Operador IEC: módulo da divisão de uma variável por outra.

Tipos permitidos: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT e ULINT. O resultado desta função será o resto da divisão. O resultado será um número inteiro.

Exemplo em IL:

<b>LD</b>		9	
<b>MOD</b>		2	
<b>ST</b>		Var1	

O resultado em Var1 é 1.

Exemplo em ST:

```
var1 := 9 MOD 2;
```

Exemplo em FBD:



**MOVE**

Operador IEC: atribuição de uma variável à outra (variável) de um tipo apropriado.

Uma vez que MOVE está disponível como uma caixa nos editores gráficos FBD, LD e CFC, então, nestes, a funcionalidade EN/ENO (habilitação) pode também ser aplicada em uma atribuição de variável.



**Figura 4-9. Exemplo em CFC em Combinação com EN/ENO**

Somente se en\_i for TRUE, var1 será atribuída a var2.

Exemplo em IL:

```
LD      var1
MOVE
ST      var2
```

Resultado: var2 obtém o valor de var1.

O mesmo resultado será obtido conforme o exemplo abaixo:

```
LD      var1
ST      var2
```

Exemplo em ST:

```
ivar2 := MOVE(ivar1); (* O mesmo resultado será obtido com: ivar2 :=
ivar1; *)
```

**SIZEOF**

Este operador aritmético não está prescrito na norma IEC 61131-3.

Ele pode ser usado para determinar o número de bytes requisitado pela variável x dada.

O operador SIZEOF retorna um valor não atribuído. O tipo do valor de retorno será adaptado ao tamanho encontrado da variável x.

Valor de retorno de SIZEOF(x)	Tipo de dado da constante implicitamente usada para o tamanho encontrado
0 <= size of x < 256	USINT
256 <= size of x < 65536	UINT
65536 <= size of x < 4294967296	UDINT
4294967296 <= size of x	ULINT

**Tabela 4-11. Operador SIZEOF**

Exemplo em ST:

```
VAR
arr1:ARRAY[0..4] OF INT;
Var1:INT;
end_var
Var1 := SIZEOF(arr1); (* d.h.: var1:=USINT#10; *)
```

Exemplo em IL:

<b>LD</b>	arr1	
<b>SIZEOF</b>		
<b>ST</b>	Var1	

O resultado é 10.

### INDEXOF

Este operador aritmético não está prescrito na norma IEC 61131-3.

Execute esta função para a classificação interna da POU.

Exemplo em ST:

```
var1 := INDEXOF(POU2);
```

### Operadores de Bitstring

Os seguintes operadores de bitstring, correspondentes à norma IEC 61131, estão disponíveis:

AND, OR, XOR e NOT.

Operadores de bitstring comparam os bits correspondentes de dois ou mais operandos.

### AND

Operador de bitstring IEC: bitwise AND de operandos de bit . Se os bits de entrada forem “1”, o bit resultante será “1”. Caso contrário, será “0”.

Tipos permitidos: BOOL, BYTE, WORD, DWORD e LWORD.

Exemplo em IL:

<b>LD</b>	2#1001_0011	
<b>AND</b>	2#1000_1010	
<b>ST</b>	var1	

O resultado em “Var1” é 2#1000\_0010.

Exemplo em ST:

```
VAR
Var1:BYTE;
END_VAR
var1 := 2#1001_0011 AND 2#1000_1010;
```

Exemplo em FBD:



### OR

Operador de bitstring IEC: OR bitwise de operandos de bit. Se no mínimo um dos bits de entrada for “1”, o bit resultante será “1”. Caso contrário, será “0”.

Tipos permitidos: BOOL, BYTE, WORD ou DWORD e LWORD.

Exemplo em IL:

<b>LD</b>	2#1001_0011	
<b>OR</b>	2#1000_1010	
<b>ST</b>	Var1	

O resultado em “var1” (do tipo BYTE) é 2#1001\_1011.

Exemplo em ST:

```
Var1 := 2#1001_0011 OR 2#1000_1010;
```

Exemplo em FBD:



## XOR

Operador de bitstring IEC: operação XOR bitwise de operandos de bit. Se apenas um dos bits de entrada for “1”, o bit resultante será “1”. Se ambos ou nenhum forem “1”, o bit resultante será “0”.

Tipos permitidos: BOOL, BYTE, WORD, DWORD e LWORD.

NOTA: Observe o comportamento da função XOR na forma estendida (se houver mais de 2 entradas). As entradas serão verificadas em pares e os resultados específicos serão, então, comparados novamente em pares (isto está em conformidade com a norma, mas pode não ser esperado pelo usuário).

Exemplo em IL:

<b>LD</b>	2#1001_0011	
<b>XOR</b>	2#1000_1010	
<b>ST</b>	var1	

O resultado em “var1” (do tipo BYTE) é 2#0001\_1001.

Exemplo em ST:

```
Var1 := 2#1001_0011 XOR 2#1000_1010;
```

Exemplo em FBD:



## NOT

Operador de bitstring IEC: operação NOT bitwise de um operando de bit. O bit resultante será “1” se o bit de entrada correspondente for “0” e vice versa.

Tipos permitidos: BOOL, BYTE, WORD, DWORD e LWORD.

Exemplo em IL:

<b>LD</b>	2#1001_0011	
<b>NOT</b>		
<b>ST</b>	var1	

O resultado em “var1” (do tipo BYTE) é 2#0110\_1100.

Exemplo em ST:

```
Var1 := NOT 2#1001_0011;
```

Exemplo em FBD:



## Operadores de Deslocamento de Bits

Os seguintes operadores de deslocamento de bits, correspondentes à norma IEC 61131, estão disponíveis:

SHL, SHR, ROL e ROR.

### SHL

Operador IEC: deslocamento bitwise de um operando à esquerda.

erg:= SHL (in, n)

in: operando a ser deslocado para a esquerda.

n: número de bits pelos quais in é deslocado para a esquerda.

Se n exceder o tamanho do tipo de dado, os operandos do tipo BYTE, WORD, DWORD e LWORD serão preenchidos com zeros. Os operandos de tipos de dados atribuídos, como por exemplo INT, sofrerão um deslocamento aritmético, ou seja, eles serão preenchidos com o valor do bit mais acima.

#### NOTAS:

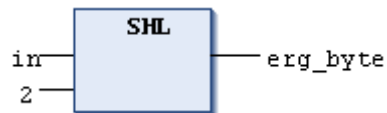
- Observe que a quantidade de bits necessária para a operação aritmética deve ser aquela pretendida pelo tipo de dado da variável de entrada. Se a variável de entrada for uma constante, será considerado o menor tipo de dado possível. O tipo de dado da variável de saída não tem efeito na operação aritmética.
- Veja no exemplo seguinte, em notação hexadecimal, que diferentes resultados podem ser obtidos para “erg\_byte” e “erg\_word” dependendo do tipo de dado da variável de entrada (BYTE ou WORD), embora os valores das variáveis de entrada in\_byte e in\_word sejam os mesmos.

Exemplo em ST:

```

PROGRAM shl_st
VAR
in_byte : BYTE:=16#45;
in_word : WORD:=16#45;
erg_byte : BYTE;
erg_word : WORD;
n: BYTE :=2;
END_VAR
erg_byte:=SHL(in_byte,n); (* O resultado é 16#14 *)
erg_word:=SHL(in_word,n); (* O resultado é 16#0114 *)
  
```

Exemplo em FBD:



Exemplo em IL:

<b>LD</b>		in_byte	
<b>SHL</b>		2	
<b>ST</b>		erg_byte	

**SHR**

Operador: deslocamento bitwise à direita de um operando.

`erg:= SHR (in, n)`

`in`: operando a ser deslocado para a direita.

`n`: número de bits pelos quais `in` é deslocado para a direita.

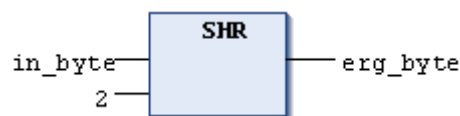
Se `n` exceder o tamanho do tipo de dado, os operandos do tipo **BYTE**, **WORD**, **DWORD** e **LWORD** serão preenchidos com zeros. Os operandos de tipos de dados atribuídos, como por exemplo **INT**, obterão um deslocamento aritmético, ou seja, eles serão preenchidos com o valor do próximo bit.

**NOTAS:**

- Observe que a quantidade de bits necessária para a operação aritmética deve ser aquela pretendida pelo tipo de dado da variável de entrada. Se a variável de entrada for uma constante, será considerado o menor tipo de dado possível. O tipo de dado da variável de saída não tem efeito na operação aritmética.
- Veja o seguinte exemplo, em notação hexadecimal, para verificar os resultados da operação aritmética dependendo do tipo da variável de entrada (**BYTE** ou **WORD**).

**Exemplo em ST:**

```
PROGRAM shr_st
VAR
in_byte : BYTE:=16#45;
in_word : WORD:=16#45;
erg_byte : BYTE;
erg_word : WORD;
n: BYTE :=2;
END_VAR
erg_byte:=SHR(in_byte,n); (* O resultado é 16#11 *)
erg_word:=SHR(in_word,n); (* O resultado é 16#0011 *)
```

**Exemplo em FBD:****Exemplo em IL:**

<b>LD</b>		<code>in_byte</code>	
<b>SHR</b>		<code>2</code>	
<b>ST</b>		<code>erg_byte</code>	

**ROL**

Operador IEC: rotação bitwise de um operando para a esquerda.

`erg:= ROL (in, n)`

Tipos de dados permitidos: **BYTE**, **WORD**, **DWORD** e **LWORD**.

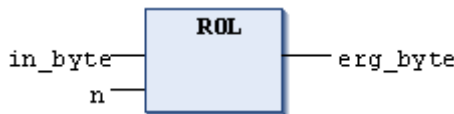
`in` será deslocado uma posição de bit para a esquerda `n` vezes, enquanto o bit que está mais à esquerda será reinserido a partir da direita.

**NOTAS:**

- Observe que a quantidade de bits necessária para a operação aritmética deve ser aquela pretendida pelo tipo de dado da variável de entrada. Se a variável de entrada for uma constante, será considerado o menor tipo de dado possível. O tipo de dado da variável de saída não tem efeito na operação aritmética.
- Veja no exemplo seguinte, em notação hexadecimal, que diferentes resultados podem ser obtidos para “erg\_byte” e “erg\_word” dependendo do tipo de dado da variável de entrada (BYTE ou WORD), embora os valores das variáveis de entrada “in\_byte” e “in\_word” sejam os mesmos.

**Exemplo em ST:**

```
PROGRAM rol_st
VAR
in_byte : BYTE:=16#45;
in_word : WORD:=16#45;
erg_byte : BYTE;
erg_word : WORD;
n: BYTE :=2;
END_VAR
erg_byte:=ROL(in_byte,n); (* O resultado é 16#15 *)
erg_word:=ROL(in_word,n); (* O resultado é 16#0114 *)
```

**Exemplo em FBD:****Exemplo em IL:**

<b>LD</b>	in_byte		
<b>ROL</b>	n		
<b>ST</b>	erg_byte		

**ROR**

Operador IEC: rotação de bitwise de um operando para a direita.

erg = ROR (in, n)

Tipos de dados permitidos: BYTE, WORD, DWORD e LWORD.

In será deslocado uma posição de bit para a direita n vezes, enquanto o bit que está mais a esquerda será reinserido a partir da esquerda.

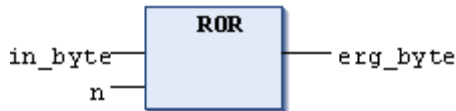
**NOTAS:**

- Observe que a quantidade de bits necessária para a operação aritmética deve ser aquela pretendida pelo tipo de dado da variável de entrada. Se a variável de entrada for uma constante, será considerado o menor tipo de dado possível. O tipo de dado da variável de saída não tem efeito na operação aritmética.
- Veja no exemplo seguinte, em notação hexadecimal, que diferentes resultados podem ser obtidos para “erg\_byte” e “erg\_word” dependendo do tipo de dado da variável de entrada (BYTE ou WORD), embora os valores das variáveis de entrada “in\_byte” e “in\_word” sejam os mesmos.

Exemplo em ST:

```
PROGRAM ror_st
VAR
in_byte : BYTE:=16#45;
in_word : WORD:=16#45;
erg_byte : BYTE;
erg_word : WORD;
n: BYTE :=2;
END_VAR
erg_byte:=ROR(in_byte,n); (* O resultado é 16#51 *)
erg_word:=ROR(in_word,n); (* O resultado é 16#4011 *)
```

Exemplo em FBD:



Exemplo em IL:

<b>LD</b>		in_byte	
<b>ROR</b>		n	
<b>ST</b>		erg_byte	

## Operadores de Seleção

Todas as operações de seleção podem também ser executadas com variáveis. Por questões de clareza, os exemplos são limitados àquelas operações que usam constantes como operadores: SEL, MAX, MIN, LIMIT e MUX.

### SEL

Operador de seleção IEC: seleção binária. G determina se IN0 ou IN1 serão atribuídos a OUT.

```
OUT := SEL(G, IN0, IN1)
OUT := IN0; (*Se G=FALSE*)
OUT := IN1; (*Se G=TRUE*)
```

Tipos de dados permitidos:

IN0, IN1 e OUT: Qualquer Tipo;

G: BOOL;

Exemplo em IL:

<b>LD</b>		TRUE	
<b>SEL</b>		3	,
		4	
<b>ST</b>		Var1	

O resultado é 4.

<b>LD</b>		FALSE	
<b>SEL</b>		3	,
		4	
<b>ST</b>		Var1	

O resultado é 3.



Exemplo em ST:

```
Var1:=SEL(TRUE,3,4); (* O resultado é 4 *)
```

Exemplo em FBD:



NOTA: Observe que uma expressão em IN1 ou IN2 não será processada se IN0 for TRUE.

## MAX

Operador de seleção IEC: função máximo. Retorna o maior dos dois valores.

```
OUT := MAX(IN0, IN1);
```

IN0, IN1 e OUT podem ser de qualquer tipo de variável.

Exemplo em IL:

LD	90		
MAX	30		
MAX	40		
MAX	77		
ST	Var1		

O resultado é 90.

Exemplo em ST:

```
Var1:=MAX(30,40); (* O resultado é 40 *)
```

```
Var1:=MAX(40,MAX(90,30)); (* O resultado é 90 *)
```

Exemplo em FBD:



## MIN

Operador de seleção IEC: função mínimo. Retorna o menor dos dois valores.

```
OUT := MIN(IN0, IN1)
```

IN0, IN1 e OUT podem ser de qualquer tipo de variável.

Exemplo em IL:

LD	90		
MIN	30		
MIN	40		
MIN	77		
ST	Var1		

O resultado é 30.

Exemplo em ST:

```
Var1:=MIN(90,30); (* O resultado é 30 *)
```

```
Var1:=MIN(MIN(90,30),40); (* O resultado é 30 *)
```

Exemplo em FBD:



## LIMIT

Operador de seleção IEC: limites.

```
OUT := LIMIT(Min, IN, Max)
```

Que pode ser escrito como:

```
OUT := MIN (MAX (IN, Min), Max);
```

Max e Min representam, respectivamente, os limites superior e inferior para o resultado. Se o valor IN exceder o limite superior (Max), LIMIT retornará Max. Se IN for inferior a Min, o resultado será Min.

IN e OUT podem ser de qualquer tipo de variável.

Exemplo em IL:

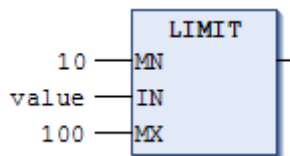
<b>LD</b>	90		
<b>LIMIT</b>	30	,	
	80		
<b>ST</b>	Var1		

O resultado é 80.

Exemplo em ST:

```
Var1:=LIMIT(30,90,80); (* O resultado é 80 *)
```

Exemplo em FBD:



## MUX

Operador de seleção IEC: operador de multiplexação.

```
OUT := MUX(K, IN0, ..., INn)
```

Que pode ser escrito como:

```
OUT := INK;
```

IN0, ..., INn e OUT podem ser de qualquer tipo de variável. K deve ser BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, LINT, ULINT ou UDINT. MUX seleciona o valor "K" dentre um grupo de valores.

Exemplo em IL:

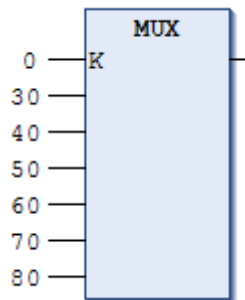
<b>LD</b>	0	
<b>MUX</b>	30	r
	40	r
	50	r
	60	r
	70	r
	80	
<b>ST</b>	Var1	

O resultado é 30.

Exemplo em ST:

Var1 := MUX(0,30,40,50,60,70,80); (\* O resultado é 30 \*)

Exemplo em FBD:



O resultado é 30.

NOTA: Uma expressão em outra entrada (que não seja K) não será processada para otimizar o tempo de execução. Somente no modo de simulação todas as expressões serão executadas.

## Operadores de Comparação

Os seguintes operadores, compatíveis com a norma IEC 61131, estão disponíveis:

GT, LT, LE, GE, EQ e NE.

Estes são operadores booleanos, cada qual comparando duas entradas (primeiro e segundo operando).

### GT

Operador de comparação IEC: maior que.

Trata-se de um operador booleano que retorna o valor TRUE quando o valor do primeiro operando for maior que o segundo. Os operandos podem ser de qualquer tipo de dado numérico básico.

Exemplo em IL:

<b>LD</b>	20	
<b>GT</b>	30	
<b>ST</b>	Var1	

O resultado é FALSE.

Exemplo em ST:

VAR1 := 20 > 30 > 40 > 50 > 60 > 70;

Exemplo em FBD:



**LT**

Operador de comparação IEC: menor que.

Trata-se de um operador booleano que retorna o valor TRUE quando o valor do primeiro operando for menor que o segundo. Os operandos podem ser de qualquer tipo de dado básico.

Exemplo em IL:

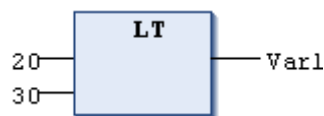
<b>LD</b>	20	
<b>LT</b>	30	
<b>ST</b>	Var1	

O resultado é TRUE.

Exemplo em ST:

```
VAR1 := 20 < 30;
```

Exemplo em FBD:



**LE**

Operador de comparação IEC: menor ou igual a.

Um operador booleano retorna o valor TRUE quando o valor do primeiro operando for menor ou igual ao segundo. Os operandos podem ser de qualquer tipo de dado numérico básico.

Exemplo em IL:

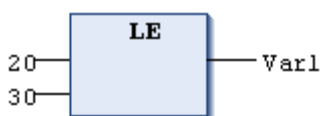
<b>LD</b>	20	
<b>LE</b>	30	
<b>ST</b>	Var1	

O resultado é TRUE.

Exemplo em ST:

```
VAR1 := 20 <= 30;
```

Exemplo em FBD:



**GE**

Operador de comparação IEC: maior ou igual a.

Trata-se de um operador booleano que retorna o valor TRUE quando o valor do primeiro operando for maior ou igual ao valor do segundo. Os operandos podem ser de qualquer tipo de dado numérico básico.

Exemplo em IL:

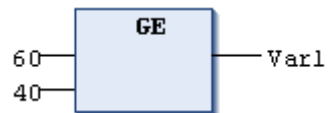
<b>LD</b>	60		
<b>GE</b>	40		
<b>ST</b>	Var1		

O resultado é TRUE.

Exemplo em ST:

VAR1 := 60 >= 40;

Exemplo em FBD:



## EQ

Operador de comparação IEC: igual a.

Trata-se de um operador booleano que retorna o valor TRUE quando os operandos comparados são iguais. Os operandos podem ser de qualquer tipo de dado numérico básico.

Exemplo em IL:

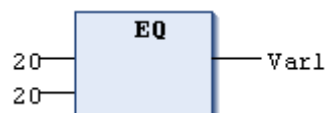
<b>LD</b>	40		
<b>EQ</b>	40		
<b>ST</b>	Var1		

O resultado é TRUE.

Exemplo em ST:

VAR1 := 40 = 40;

Exemplo em FBD:



## NE

Operador de comparação IEC: diferente de.

Trata-se de um operador booleano que retorna o valor TRUE quando os operandos não são iguais. Os operandos podem ser de qualquer tipo de dado básico.

Exemplo em IL:

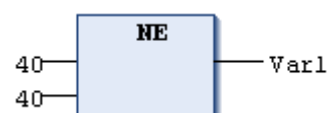
<b>LD</b>	40		
<b>NE</b>	40		
<b>ST</b>	Var1		

O resultado é FALSE.

Exemplo em ST:

VAR1 := 40 <> 40;

Exemplo em FBD:



## Operadores de Endereço

ADR, BITADR e o operador de conteúdo “^” são operadores de endereços extensivos à norma disponíveis no MasterTool IEC XE.

### ADR

Este operador de endereço não está prescrito na norma IEC 61131-3.

ADR retorna o endereço de seu argumento em uma DWORD. Este endereço pode ser enviado para funções, podendo ser tratado como um ponteiro, ou pode ser atribuído a um ponteiro em um projeto.

NOTA: O operador ADR pode ser usado com nomes de funções, programas, blocos funcionais e métodos, assim substituindo o operador INDEXOF.

Considere que ponteiros de função podem ser passados a bibliotecas externas, mas não há possibilidade de chamar um ponteiro de função no MasterTool IEC XE. Para habilitar uma chamada de sistema (sistema de execução) a respectiva propriedade (categoria *Compile*) deve estar definida para a função. Consulte **Ponteiros de Função**.

Exemplo em ST:

```
dwVar:=ADR(bVar);
```

Exemplo em IL:

LD	bVar		
ADR			
ST	dwVar		

NOTA: Após uma alteração online, podem ocorrer mudanças em relação aos dados em alguns endereços. Isto deve ser observado ao usar ponteiros em endereços.

### BITADR

Este operador de endereço não está prescrito na norma IEC 61131-3.

BITADR retorna o offset do bit no segmento em uma DWORD. Observe que o valor do offset depende da opção de endereçamento de byte nas configurações do dispositivo estar ativada ou não.

```
VAR
var1 AT %IX2.3:BOOL;
bitoffset: DWORD;
END_VAR
```

Exemplo em ST:

```
bitoffset:=BITADR(var1); (* Resultado: 16#80000013*)
```

Exemplo em IL:

LD	Var1		
BITADR			
ST	bitoffset		

NOTA: Após uma alteração online, podem ocorrer mudanças em relação aos dados em alguns endereços. Isto deve ser observado ao usar ponteiros em endereços.

### Operador de Conteúdo

Este operador de endereço não está prescrito na norma IEC 61131-3. Um ponteiro pode ser desreferenciado através da adição do operador de conteúdo “^” após o identificador do ponteiro.

Exemplo em ST:

```
pt:POINTER TO INT;
var_int1:INT;
var_int2:INT;
pt := ADR(var_int1);
var_int2:=pt^;
```

NOTA: Após uma alteração online, podem ocorrer mudanças em relação aos dados em alguns endereços. Isto deve ser observado ao usar ponteiros em endereços.

## Operador de Chamada

### CAL

Operador IEC para chamar um bloco funcional ou um programa.

Use CAL em IL para chamar uma instância de bloco funcional. As variáveis que servirão como variáveis de entrada estão localizadas entre parênteses imediatamente à direita do nome da instância do bloco funcional.

Exemplo:

Chamando a instância “Inst” de um bloco funcional onde as variáveis de entrada Par1 e Par2 são 0 e TRUE, respectivamente.

```
CAL INST (PAR1 := 0, PAR2 := TRUE)
```

## Funções de Conversão de Tipo

A conversão de um tipo maior em um menor (por exemplo de INT para BYTE ou de DINT para WORD) é proibida. Pode-se, basicamente, converter qualquer tipo elementar em outro tipo elementar.

Sintaxe:

```
<elem.Typ1>_TO_<elem.Typ2>
```

Observe que nas conversões ...\_TO\_STRING, a string é gerada no sentido justificado à esquerda. Se estiver definido que ela deve ser mais curta, o corte deve ser feito do lado direito.

As seguintes conversões de tipo são suportadas:

- Conversões BOOL\_TO
- Conversões TO\_BOOL
- Conversão entre tipos de números inteiros
- Conversões REAL\_TO-/ LREAL\_TO
- Conversões TIME\_TO/TIME\_OF\_DAY
- Conversões DATE\_TO/DT\_TO
- Conversões STRING\_TO
- TRUNC (conversão para DINT)
- TRUNC\_INT
- ANY\_NUM\_TO\_<numeric datatype>
- ANY\_TO\_<any datatype>

### Conversões BOOL\_TO

Operador IEC: conversão do tipo BOOL para qualquer outro tipo.

Sintaxe para um operador de conversão BOOL\_TO:

```
BOOL_TO_<tipo de dado>
```

Para tipos de números: o resultado será "1", quando o operando for TRUE. Será "0", quando o operando for FALSE.

Para tipo STRING: o resultado será TRUE, quando o operando for TRUE. Será FALSE quando o operando for FALSE.

Exemplos em IL:

<b>LD</b>	TRUE
<b>BOOL_TO_INT</b>	
<b>ST</b>	i

O resultado é 1.

<b>LD</b>	TRUE
<b>BOOL_TO_STRING</b>	
<b>ST</b>	str

O resultado é TRUE.

<b>LD</b>	TRUE
<b>BOOL_TO_TIME</b>	
<b>ST</b>	t

O resultado é T#1ms.

<b>LD</b>	TRUE
<b>BOOL_TO_TOD</b>	
<b>ST</b>	tof

O resultado é TOD#00:00:00.001.

<b>LD</b>	FALSE
<b>BOOL_TO_DATE</b>	
<b>ST</b>	dandt

O resultado é D#1970-01-01.

<b>LD</b>	TRUE
<b>BOOL_TO_DT</b>	
<b>ST</b>	dandt

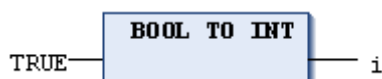
O resultado é DT#1970-01-01-00:00:01.

Exemplos em ST:

```

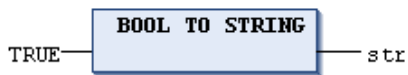
i:=BOOL_TO_INT(TRUE);           (* O resultado é 1 *)
str:=BOOL_TO_STRING(TRUE);      (* O resultado é "TRUE" *)
t:=BOOL_TO_TIME(TRUE);         (* O resultado é T#1ms *)
tof:=BOOL_TO_TOD(TRUE);        (* O resultado é
TOD#00:00:00.001 *)
dat:=BOOL_TO_DATE(FALSE);      (* O resultado é D#1970-01-
01 *)
dandt:=BOOL_TO_DT(TRUE);       (* O resultado é DT#1970-01-
01-00:00:01 *)
    
```

Exemplos em FBD:

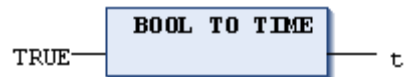


O resultado é 1.

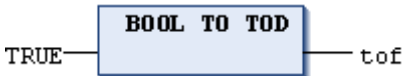




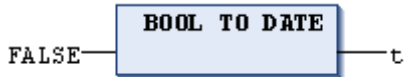
O resultado é TRUE.



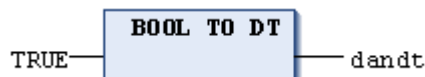
O resultado é T#1ms.



O resultado é TOD#00:00:00.001.



O resultado é D#1970-01-01.



O resultado é DT#1970-01-01-00:00:01.

### Conversões TO\_BOOL

Operador IEC: conversão de outro tipo de variável para BOOL.

Sintaxe para um operador de conversão TO\_BOOL:

<Tipo de dado>\_TO\_BOOL

O resultado é TRUE quando o operando for diferente de 0 (zero). O resultado é FALSE quando o operando for 0 (zero).

O resultado é TRUE para variáveis do tipo STRING quando o operando for TRUE caso contrário, o resultado será FALSE.

Exemplos em IL:

<b>LD</b>		213
<b>BYTE_TO_BOOL</b>		
<b>ST</b>		b

O resultado é TRUE.

<b>LD</b>		0
<b>INT_TO_BOOL</b>		
<b>ST</b>		b

O resultado é FALSE.

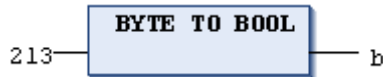
<b>LD</b>		T#5ms
<b>TIME_TO_BOOL</b>		
<b>ST</b>		b

O resultado é TRUE.

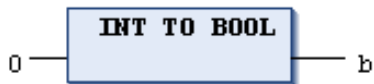
<b>LD</b>		'TRUE'
<b>STRING_TO_BOOL</b>		
<b>ST</b>		b

O resultado é TRUE.

Exemplos em FBD:



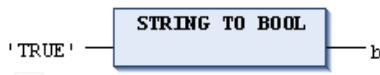
O resultado é TRUE.



O resultado é FALSE.



O resultado é TRUE.



O resultado é TRUE.

Exemplos em ST:

```

b :=
  BYTE_TO_BOOL(2#11010101);      (* O resultado é
                                  TRUE *)

b := INT_TO_BOOL(0);             (* O resultado é
                                  FALSE *)

b := TIME_TO_BOOL(T#5ms);       (* O resultado é
                                  TRUE *)

b :=
  STRING_TO_BOOL('TRUE');        (* O resultado é
                                  TRUE *)

```

### Conversão entre Tipos de Números Inteiros

Conversão de um tipo de número inteiro em outro tipo.

Sintaxe para o operador de conversão:

```
<Tipo de dado INT>_TO_<tipo de dado INT>
```

Ao executar uma conversão de um tipo maior para um menor, há o risco de perder algumas informações. Se o número a ser convertido exceder o limite do intervalo, os primeiros bytes do número serão ignorados.

Exemplo em ST:

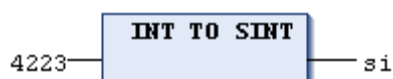
```
si := INT_TO_SINT(4223); (* O resultado é 127 *)
```

Ao salvar o inteiro 4223 (16#107F representado hexadecimalmente) como uma variável SINT, o mesmo aparecerá como 127 (16#7F representado hexadecimalmente).

Exemplo em IL:

<b>LD</b>		4223
<b>INT_TO_SINT</b>		
<b>ST</b>		si

Exemplo em FBD:



### Conversões REAL\_TO / LREAL\_TO

Operador IEC: conversão da variável tipo REAL ou LREAL para um tipo diferente.

O valor será arredondado para cima ou para baixo do número inteiro mais próximo e convertido em um novo tipo de variável. As exceções desta regra são os tipos de variáveis STRING, BOOL, REAL e LREAL.

NOTA: Se um REAL ou LREAL for convertido em SINT, USINT, INT, UINT, DINT, UDINT, LINT ou ULINT e o valor do número real estiver fora da faixa de valor daquele inteiro, o resultado será indefinido e dependerá do dispositivo. Até mesmo uma exceção é possível neste caso. Para obter um código independente do dispositivo, a parte que exceder a faixa deve ser tratada através da aplicação. Se o número REAL/LREAL estiver dentro da faixa de valor inteiro, a conversão funcionará em todos os dispositivos da mesma forma.

Em uma conversão para o tipo STRING, o número total de dígitos é limitado a 16. Caso o número REAL/LREAL apresente mais dígitos, o décimo sexto será arredondado. Se o comprimento da STRING for curto, esta será cortada a partir da extremidade direita.

Ao executar uma conversão de um tipo maior para um tipo menor, há o risco de perder algumas informações.

Exemplo em ST:

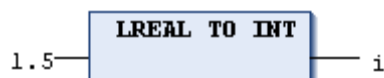
```
i := REAL_TO_INT(1.5); (* O resultado é 2 *)
j := REAL_TO_INT(1.4); (* O resultado é 1 *)
i := REAL_TO_INT(-1.5); (* O resultado é -2 *)
j := REAL_TO_INT(-1.4); (* O resultado é -1 *)
```

Exemplo em IL:

<b>LD</b>	2.7
<b>REAL_TO_INT</b>	
<b>ST</b>	i

O resultado é 3.

Exemplo em FBD:



### Conversões TIME\_TO/TIME\_OF\_DAY

Operador IEC: conversão do tipo de variável TIME ou TIME\_OF\_DAY para um tipo diferente.

Sintaxe para o operador de conversão:

<Tipo de dado TIME>\_TO\_<tipo de dado>

O tempo será armazenado internamente em uma DWORD em milissegundos (iniciando com 12:00 A.M. em uma variável TIME\_OF\_DAY). Este valor será, então, convertido.

A conversão de um tipo maior em um menor pode implicar na perda de algumas informações.

Para o tipo de variável STRING, o resultado é uma constante de tempo.

Exemplos em IL:

<b>LD</b>	T#12ms
<b>TIME_TO_STTI...</b>	
<b>ST</b>	str

O resultado é T#12ms.

<b>LD</b>	T#300000ms
<b>TIME_TO_DWORD</b>	
<b>ST</b>	dw

O resultado é 300000.

<b>LD</b>	TOD#00:00:00.012
<b>TIME_TO_SINT</b>	
<b>ST</b>	si

O resultado é 12.

Exemplos em ST:

```

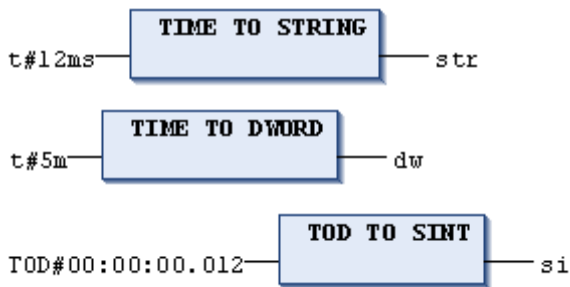
str :=TIME_TO_STRING(T#12ms);           (* O resultado é
                                         T#12ms *)

dw:=TIME_TO_DWORD(T#5m);                (* O resultado é
                                         300000 *)

si:=TOD_TO_SINT(TOD#00:00:00.012);     (* O resultado é
                                         12 *)

```

Exemplos em FBD:



### Conversões DATE\_TO/DT\_TO

Operador IEC: conversão do tipo de variável DATE ou DATE\_AND\_TIME para um tipo diferente.

Sintaxe para o operador de conversão:

<Tipo de dado DATE>\_TO\_<tipo de dado>

A data será armazenada internamente em uma DWORD, em segundos, a partir de 1 de Janeiro de 1970. Este valor será, então, convertido.

Ao executar uma conversão de um tipo maior para um menor, há o risco de perder algumas informações.

Para variáveis do tipo STRING, o resultado é uma constante de data.

Exemplos em IL:

<b>LD</b>	D#1970-01-01
<b>DATE_TO_BOOL</b>	
<b>ST</b>	b

O resultado é FALSE.

<b>LD</b>	D#1970-01-01
<b>DATE_TO_INT</b>	
<b>ST</b>	i

O resultado é 29952.

<b>LD</b>	D#1970-01-15-05:05:
<b>DATE_TO_BYTE</b>	
<b>ST</b>	byt

O resultado é 129.

<b>LD</b>	D#1998-02-13-14:20
<b>DATE_TO_STRI..</b>	
<b>ST</b>	str

O resultado é DT#1998-02-13-14:20.

Exemplos em ST:

```

b :=DATE_TO_BOOL(D#1970-01-01);           (*O resultado é
                                           FALSE*)

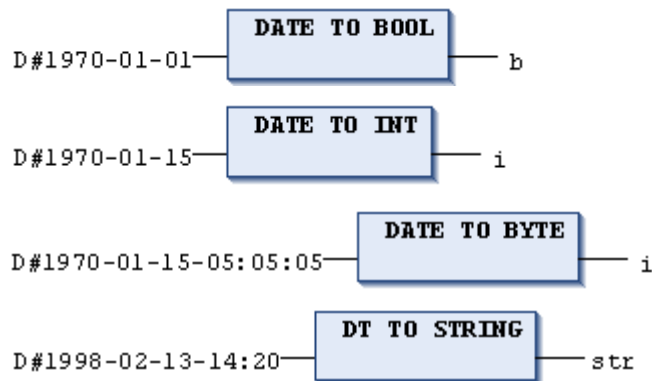
i :=DATE_TO_INT(D#1970-01-15);           (*O resultado é
                                           29952*)

byt :=DT_TO_BYTE(DT#1970-01-15-
05:05:05);                               (*O resultado é
                                           129*)

str:=DT_TO_STRING(DT#1998-02-13-
14:20);                                   (*O resultado é
                                           DT#1998-02-13-
                                           14:20*)

```

Exemplos em FBD:



### Conversões *STRING\_TO*

Operador IEC: conversão do tipo de variável *STRING* para um tipo diferente.

A conversão trabalha de acordo com o mecanismo de compilação C standard: *STRING* para *INT* e, a seguir, *INT* para *BYTE*. O byte maior será cortado e, assim, somente resultarão valores entre 0-255.

Desta forma, é possível, para a maioria dos processadores, gerar um código ótimo. A conversão pode ser executada através de uma instrução de máquina simples.

Sintaxe para o operador de conversão:

```
STRING_TO_<tipo de dado>
```

O operando da variável do tipo *STRING* deve conter um valor válido do tipo de variável do dispositivo. Caso contrário, o resultado será 0.

<b>LD</b>	'TRUE'
<b>STRING_TO_BOOL</b>	
<b>ST</b>	b

O resultado é TRUE.

<b>LD</b>	'abc34'
<b>STRING_TO_WORD</b>	
<b>ST</b>	w

O resultado é 0.

<b>LD</b>	't#117ms'
<b>STRING_TO_TIME</b>	
<b>ST</b>	t

O resultado é T#117ms.

<b>LD</b>	'500'
<b>STRING_TO_BYTE</b>	
<b>ST</b>	bv

O resultado é 244.

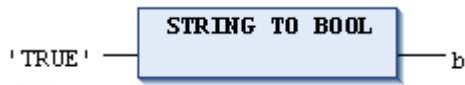
Exemplos em ST:

```

b :=STRING_TO_BOOL('TRUE');          (* O resultado é TRUE
*)
w :=STRING_TO_WORD('abc34');         (* O resultado é 0 *)
t
:=STRING_TO_TIME('T#127ms');        (* O resultado é
T#127ms *)
bv :=STRING_TO_BYTE('500');          (* O resultado é 244
*)

```

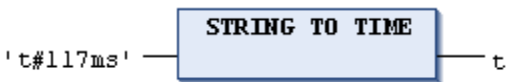
Exemplos em FBD:



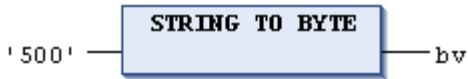
O resultado é TRUE.



O resultado é 0.



O resultado é T#127ms.



O resultado é 244.

## TRUNC

Operador IEC: conversão de REAL para DINT. A parte inteira do número do valor será usada.

NOTA: O operador TRUNC converte de REAL para INT, no entanto, TRUNC automaticamente será substituído por TRUNC\_INT.

A conversão de um tipo maior em um menor pode implicar na perda de algumas informações.

Exemplo em IL:

<b>LD</b>	1.9		
<b>TRUNC</b>			
<b>ST</b>	diVar		

Exemplos em ST:

```
diVar:=TRUNC(1.9); (* O resultado é 1 *)
diVar:=TRUNC(-1.4); (* O resultado é -1 *)
```

### TRUNC\_INT

Operador IEC: conversão de REAL para INT. A parte inteira do valor será usada.

NOTA: TRUNC_INT corresponde ao operador TRUNC tradicional.
--

Ao executar uma conversão de um tipo maior em um menor, há um risco de perda de informações.

Exemplo em IL:

<b>LD</b>	1.9		
<b>TRUNC_INT</b>			
<b>ST</b>	iVar		

Exemplos em ST:

```
iVar:=TRUNC_INT(1.9); (* O resultado é 1 *)
iVar:=TRUNC_INT(-1.4); (* O resultado é -1 *)
```

### Conversões ANY...TO

Um operador IEC específico pode ser utilizado na conversão de qualquer tipo de dado (mais especificamente, a partir de um tipo de dado numérico para outro tipo de dado). Os tipos de dados mais adequados são apresentados nas conversões mostradas a seguir:

Sintaxe:

```
ANY_NUM_TO_<Tipo de dado numérico>
ANY_TO_<Qualquer tipo de dado>
```

Exemplo:

Conversão de uma variável do tipo REAL para INT:

```
re : REAL := 1.234;
i : INT := ANY_TO_INT(re); (* O resultado é 1 *)
```

### Funções Numéricas

Os seguintes operadores numéricos IEC estão disponíveis:

ABS, SQRT, LN, LOG, EXP, SIN, COS, TAN, ASIN, ACOS, ATAN e EXPT.

#### ABS

Operador IEC: retorna o valor absoluto de um número. ABS(-2) retorna 2.

A entrada e saída podem ser de qualquer tipo de dado numérico básico.

Exemplo em IL:

<b>LD</b>	-2		
<b>ABS</b>			
<b>ST</b>	i		

O resultado em “i” é 2.

Exemplo em ST:

```
i := ABS (-2) ;
```

Exemplo em FBD:



## SQRT

Operador IEC: retorna a raiz quadrada de um número.

A variável de entrada pode ser de qualquer tipo de dado numérico básico. A variável de saída deve ser do tipo REAL ou LREAL.

Exemplo em IL:

<b>LD</b>	16		
<b>SQRT</b>			
<b>ST</b>	q		

O resultado em “q” é 4.

Exemplo em ST:

```
q := SQRT (16) ;
```

Exemplo em FBD:



## LN

Operador IEC: retorna o logaritmo natural de um número.

A variável de entrada pode ser de qualquer tipo de dado numérico básico. A variável de saída deve ser do tipo REAL ou LREAL.

Exemplo em IL:

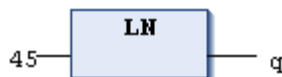
<b>LD</b>	45		
<b>LN</b>			
<b>ST</b>	q		

O resultado em “q” é 3,80666.

Exemplo em ST:

```
q := LN (45) ;
```

Exemplo em FBD:



## LOG

Operador IEC: retorna o logaritmo de um número na base 10.

A variável de entrada pode ser de qualquer tipo de dado numérico básico. A variável de saída deve ser do tipo REAL ou LREAL.

Exemplo em IL:



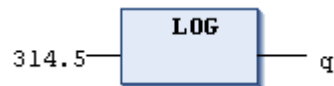
<b>LD</b>		314.5	
<b>LOG</b>			
<b>ST</b>		q	

O resultado em “q” é 2,49762.

Exemplo em ST:

q:=LOG(314.5);

Exemplo em FBD:



## EXP

Operador IEC: retorna a função exponencial.

A variável de entrada pode ser de qualquer tipo de dado numérico básico. A variável de saída deve ser do tipo REAL ou LREAL.

Exemplo em IL:

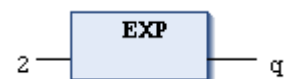
<b>LD</b>		2	
<b>EXP</b>			
<b>ST</b>		q	

O resultado em “q” é 7,389056.

Exemplo em ST:

q:=EXP(2);

Exemplo em FBD:



## SIN

Operador IEC: retorna o seno de um número.

A variável de entrada pode ser de qualquer tipo de dado numérico básico, sendo que este representa um arco em radianos. A variável de saída deve ser do tipo REAL ou LREAL.

Exemplo em IL:

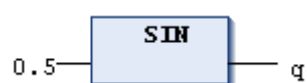
<b>LD</b>		0.5	
<b>SIN</b>			
<b>ST</b>		q	

O resultado em “q” é 0,479426.

Exemplo em ST:

q:=SIN(0.5);

Exemplo em FBD:



## COS

Operador IEC: retorna o cosseno do número.

A variável de entrada pode ser de qualquer tipo de dado numérico básico, sendo que este representa um arco em radianos. A variável de saída deve ser do tipo REAL ou LREAL.

Exemplo em IL:

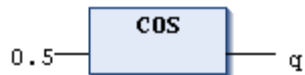
<b>LD</b>	0.5		
<b>COS</b>			
<b>ST</b>	q		

O resultado em “q” é 0.877583.

Exemplo em ST:

q:=COS(0.5);

Exemplo em FBD:



## TAN

Operador IEC: retorna a tangente de um número.

A variável de entrada pode ser de qualquer tipo de dado numérico básico, sendo que este representa um arco em radianos. A variável de saída deve ser do tipo REAL ou LREAL.

Exemplo em IL:

<b>LD</b>	0.5		
<b>TAN</b>			
<b>ST</b>	q		

O resultado em “q” é 0,546303.

Exemplo em ST:

q:=TAN(0.5);

Exemplo em FBD:



## ASIN

Operador IEC: retorna o arco seno (função inversa do seno) de um número.

A variável de entrada pode ser de qualquer tipo de dado numérico básico. A variável de saída deve ser do tipo REAL ou LREAL, sendo que esta representa um arco em radianos.

Exemplo em IL:

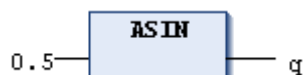
<b>LD</b>	0.5		
<b>ASIN</b>			
<b>ST</b>	q		

O resultado em “q” é 0,523599.

Exemplo em ST:

q:=ASIN(0.5);

Exemplo em FBD:



## ACOS

Operador IEC: retorna o arco cosseno (função inversa do cosseno) de um número.

A variável de entrada pode ser de qualquer tipo de dado numérico básico. A variável de saída deve ser do tipo REAL ou LREAL, sendo que esta representa um arco em radianos.

Exemplo em IL:

<b>LD</b>	0.5		
<b>ACOS</b>			
<b>ST</b>	q		

O resultado em “q” é 1,0472.

Exemplo em ST:

```
q := ACOS(0.5);
```

Exemplo em FBD:



## ATAN

Operador IEC: retorna o arco tangente (função inversa da tangente) de um número.

A variável de entrada pode ser de qualquer tipo de dado numérico básico. A variável de saída deve ser do tipo REAL ou LREAL, sendo que esta representa um arco em radianos.

Exemplo em IL:

<b>LD</b>	0.5		
<b>ATAN</b>			
<b>ST</b>	q		

O resultado em “q” é 0,463648.

Exemplo em ST:

```
q := ATAN(0.5);
```

Exemplo em FBD:



## EXPT

Operador IEC: exponenciação de uma variável com outra variável.

```
OUT = IN1IN2;
```

A variável de entrada pode ser de qualquer tipo de dado numérico básico. A variável de saída deve ser do tipo REAL ou LREAL.

Exemplo em IL: (\* O resultado é 49 \*)

<b>LD</b>	7		
<b>EXPT</b>	2		
<b>ST</b>	Var1		

Exemplo em ST:

```
var1 := EXPT(7,2);
```

Exemplo em FBD:



## Operadores Adicionais à Norma IEC

### Operadores de Escopo Adicionais à Norma IEC

Além dos operadores IEC, há várias possibilidades de precisar o acesso a variáveis ou módulos se o nome destes for usado várias vezes no escopo de um projeto. Para definir o respectivo contexto, os seguintes operadores de escopo podem ser usados:

- “.” (operador de escopo global)
- “<nome da lista de variáveis globais>”
- “<nome de biblioteca>”
- “<nome de enumeração>”

### Operador de Escopo Global

Operador de escopo: extensão da norma IEC 61131-3.

Um caminho de instância iniciando com “.” abre um escopo global (contexto). Assim, se houver uma variável local e uma global com o mesmo nome “<nome da variável>”, “.<nome da variável>” estará referindo-se à variável global.

### Nome da Lista de Variáveis Globais

Operador de escopo: extensão da norma IEC 61131-3.

O nome de uma lista de variáveis globais pode ser usado como contexto para as variáveis constantes desta lista. Assim, é possível declarar variáveis com nomes idênticos em diferentes listas e, precedendo o nome da variável por “<nome da lista de variáveis globais>.<nome da variável global>”, é possível acessar aquela desejada.

Exemplo:

As listas de variáveis globais globlist1 e globlist2 contêm uma variável chamada varx. Na linha seguinte, varx da globlist2 é copiada para varx na globlist1:

```
globlist1.varx := globlist2.varx; (* Nesta linha de código *)
```

Se for referenciado o nome de uma variável declarada em mais de uma lista de variáveis globais, sem haver o nome da lista como um operador precedente, será gerada uma mensagem de erro.

### Contexto de Biblioteca

Operador escopo: extensão da norma IEC 61131-3.

O contexto de biblioteca pode ser usado para, explicitamente, acessar os componentes da biblioteca. Exemplo: se uma biblioteca incluída em um projeto contém um módulo fun e há também uma POU fun definida localmente no projeto, o contexto da biblioteca pode ser adicionado ao nome do módulo para tornar o acesso exclusivo. A sintaxe é: “<contexto>.<nome do módulo>”, por exemplo: lib.fun.

Por padrão, o contexto de uma biblioteca é idêntico ao nome da mesma. No entanto, é possível definir outro nome, tanto nas *Informações do Projeto* (ao criar um projeto de biblioteca), quanto no diálogo *Propriedades*, posteriormente.

Exemplo:

Suponha a função fun na biblioteca lib e a função fun declarada no projeto. Por padrão, o contexto da biblioteca é nomeado lib:

```
res1 := fun(in := 12); // chamada da função do projeto FUN.  
res2 := lib.fun(in := 12); // chamada da função da biblioteca FUN.
```

### Nome da Enumeração

Operador de escopo: extensão da norma IEC 61131-3.

O nome do tipo das enumerações pode ser usado para tornar o acesso a uma constante de enumeração não ambíguo. Neste caso, “<nome da enumeração>.” precede o nome da constante. Assim, é possível usar a mesma constante em diferentes enumerações.

Exemplo:

A constante Blue é um componente das enumerações Colors e Feelings.

```
color := Colors.Blue; // Acesso ao valor de enum Blue no tipo Colors.  
feeling := Feelings.Blue; // Acesso ao valor de enum Blue no tipo  
Feelings.
```

## Operandos

Os operandos são classificados como segue:

- Constantes (BOOL, TIME, DATE, TIME\_OF\_DAY, DATE\_AND\_TIME, número, REAL/LREAL, STRING, Literais tipados)
- Variáveis
- Endereços
- Funções

### Constantes

#### Constantes BOOL

As constantes BOOL assumem os valores lógicos TRUE e FALSE.

Veja também: **BOOL (Tipos de Dados Padrão)**.

#### Constantes TIME

As constantes TIME são geralmente usadas para operar módulos de tempo padrão. Além da constante TIME, cujo tamanho é 32 bits e está em conformidade com a norma IEC 61131-3, LTIME é suportada como base de tempo para temporizadores de alta resolução. LTIME tem o tamanho de 64 bits e resolução em nanosegundos.

Sintaxe para a constante TIME:

T#<Declaração do tempo>

Em vez de “T#”, também é possível usar “t#”, “time#” e “TIME#”.

A declaração de tempo pode incluir as unidades de tempo descritas abaixo. Elas devem ser usadas na sequência a seguir, porém não é necessário utilizar todas elas:

- “d”: dias
- “h”: horas
- “m”: minutos
- “s”: segundos
- “ms”: milissegundos

Exemplos de constantes TIME corretas em uma atribuição ST:

```
TIME1 := T#14ms;  
TIME1 := T#100S12ms; (* O componente mais alto pode exceder este limite *)  
TIME1 := t#12h34m15s;
```

Abaixo, exemplos de utilização incorreta:

```
TIME1 := t#5m68s; (* Limite excedido em um componente inferior *)
TIME1 := 15ms; (* Está faltando o T# *)
TIME1 := t#4ms13d; (* Ordem incorreta dos itens *)
```

Sintaxe para a constante LTIME:

```
LTIME#<Declaração do tempo>
```

Além das unidades de tempo usadas com a constante TIME (veja acima), a declaração de tempo pode incluir:

- “us”: microssegundos
- “ns”: nanosegundos

Exemplos de constantes LTIME corretas em uma atribuição ST:

```
LTIME1 := LTIME#1000d15h23m12s34ms2us44ns
LTIME1 := LTIME#3445343m3424732874823ns
```

Veja também: **Tipos de Dado de Tempo.**

### Constantes DATE

Estas constantes podem ser usadas para inserir datas.

Sintaxe:

```
d#<Declaração de data>
```

Além de “d#”, podem ser usados também “D#”, “date#” e “DATE#”.

A declaração da data deve ser inserida no formato <ano-mês-dia>.

Valores DATE são tratados internamente como DWORD, contendo o intervalo de tempo em segundos desde 01.01.1970, 00:00 h.

Exemplos:

```
DATE#1996-05-06
d#1972-03-29
```

Veja também: **Tipos de Dado de Tempo.**

### Constantes TIME\_OF\_DAY

Este tipo de constante é utilizada para armazenar a hora do dia.

Sintaxe:

```
tod#<Declaração do tempo>
```

Em vez de “tod#”, também podem ser usados: “TOD#”, “time\_of\_day#” e “TIME\_OF\_DAY#”.

A declaração de tempo deve ser inserida no formato <hora:minuto:segundo>.

Segundos podem ser inseridos como números reais, ou seja, frações de um segundo podem ser especificadas.

Valores TIME\_OF\_DAY são tratados internamente como valores DWORD contendo o intervalo de tempo em milissegundos desde as 00:00 h.

Exemplos:

```
TIME_OF_DAY#15:36:30.123
tod#00:00:00
```

Veja também: **Tipos de Dado de Tempo.**

*Constantes DATE\_AND\_TIME*

As constantes DATE e TIME\_OF\_DAY podem também ser combinadas com as constantes DATE\_AND\_TIME.

Sintaxe:

dt#<Declaração de data e hora>

Além de “dt#”, podem ser usados também “DT#”, “date\_and\_time#” e “DATE\_AND\_TIME#”.

A declaração da data e hora deve ser inserida no formato <ano-mês-dia-hora:minuto:segundo>.

Segundos podem ser inseridos como números reais, ou seja, frações de um segundo também podem ser especificadas.

Valores DATE\_AND\_TIME são tratados internamente como DWORD contendo o intervalo de tempo em segundos desde 01.01.1970, 00:00 h.

Exemplos:

```
DATE_AND_TIME#1996-05-06-15:36:30
dt#1972-03-29-00:00:00
```

Veja também: **Tipos de Dado de Tempo.**

*Constantes Numéricas*

Valores numéricos podem assumir valores binários, octais, decimais e hexadecimais.

Se um valor inteiro não for um número decimal, a base deve ser seguida pelo sinal (#) na frente da constante inteira.

Os valores para os números 10-15 (hexadecimais) serão representados pelas letras A-F.

Pode-se incluir um sublinhado no número.

Exemplos:

```
14          (* número decimal *)
2#1001_0011 (* número binário *)
8#67        (* número octal *)
16#A        (* número hexadecimal *)
```

Estes valores podem ser dos tipos: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL ou LREAL.

Não são permitidas conversões implícitas de tipos de variável maiores para menores. Isto significa que a variável DINT não pode simplesmente ser usada como variável INT. Neste caso, deve ser usada a conversão de tipo.

*Constantes REAL/LREAL*

As constantes REAL e LREAL podem ser utilizadas como frações decimais e representadas exponencialmente. Use o formato padrão americano, com o ponto decimal.

Exemplo:

```
7.4          ( * em vez de 7,4 * )
1.64e+009    ( * em vez de 1,64e+009 * )
```

*Constantes STRING*

Uma string é uma sequência de caracteres. Constantes STRING são precedidas e seguidas por aspas simples. Também é possível inserir espaços em branco e caracteres especiais (UMLAUTS, por exemplo). Eles serão tratados da mesma forma que outros caracteres. Observe as seguintes possibilidades de usar o sinal “\$” em constantes de string:

Sinal	Resultado
<b>\$&lt;dois números hexadecimais&gt;</b>	Representação hexadecimal do código do caractere de 8 bits
<b>\$\$</b>	Sinal de dólar
<b>\$'</b>	Aspa única
<b>\$L ou \$l</b>	Avanço de linha
<b>\$N ou \$n</b>	Nova linha
<b>\$P ou \$p</b>	Alimentação da página
<b>\$R ou \$r</b>	Quebra de linha
<b>\$T ou \$t</b>	Tab

Tabela 4-12. Possibilidades de Uso para o Sinal \$

Exemplos:

```
' Abby and Craig '
':-) '
'costs ($$) '
```

### Literais Tipados

Basicamente, no uso das constantes IEC, é empregado o menor tipo de dado possível. Para usar outro tipo de dado, podem ser utilizados os chamados literais tipados, sem a necessidade de declarar as constantes explicitamente.

Para este propósito, a constante será fornecida com um prefixo que determina o seu tipo.

Sintaxe:

```
<TIPO>#<Literal>
```

<TIPO> especifica o tipo de dado desejado. Itens possíveis: BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, REAL e LREAL. O tipo deve ser escrito em letras maiúsculas.

<Literal> especifica a constante. O dado inserido deve adequar-se ao tipo dado especificado em <TIPO>.

Exemplo:

```
var1:=DINT#34;
```

Será gerada uma mensagem de erro caso a constante não possa ser convertida para o tipo do dispositivo sem perda de dados.

Literais tipados podem ser usados em qualquer local onde sejam usadas constantes normais.

### Variáveis

Variáveis podem ser declaradas localmente na parte de declaração de uma POU ou em uma lista de variáveis globais.

Consulte **Declaração de Variáveis** para obter informações sobre a declaração de uma variável, incluindo as regras referentes ao identificador da variável e seu uso múltiplo.

Variáveis podem ser usadas em qualquer local onde o tipo declarado permiti-las.

As variáveis disponíveis podem ser acessadas através do *Assistente de Entrada*.

### Acessando Variáveis

Sintaxe:

Para componentes de ARRAYS bi-dimensionais:

```
<NOME DO ARRAY>[INDICE1, INDICE2]
```



Para variáveis de estrutura:

```
<NOME DA ESTRUTURA>.<NOME DA VARIÁVEL>
```

Para blocos funcionais e variáveis de programa:

```
<NOME DO BLOCO FUNCIONAL>.<NOME DA VARIÁVEL>
```

### Endereçando Bits

Bits individuais podem ser acessados em variáveis inteiras. Para tal, o índice do bit a ser endereçado é anexado à variável, separado por um ponto. O índice do bit pode ser dado por qualquer constante. A indexação é baseada em zero.

Sintaxe:

```
<Nome da variável>.<Índice de bit>
```

Exemplo:

```
a : INT;  
b : BOOL;  
...  
a.2 := b;
```

O terceiro bit da variável A será configurado para o valor da variável B.

Se o índice for maior que o tamanho do bit de uma variável, a seguinte mensagem de erro será apresentada: “Índice '<n>' fora do intervalo válido para a variável '<var>'!”

O endereçamento de bit é possível com os seguintes tipos de variável: SINT, INT, DINT, USINT, UINT, UDINT, BYTE, WORD, DWORD.

Se o tipo da variável não permitir o acesso a bit, a seguinte mensagem de erro será gerada: “Tipo de dado inválido '<type>' para indexação direta”.

Um acesso a bit não deve ser atribuído à uma variável VAR\_IN\_OUT.

### Acesso a Bit Via Constante Global

Se foi declarada uma constante global definindo a indexação do bit, esta constante pode ser usada para um acesso a bit.

Abaixo, se encontram exemplos para um acesso a bit via uma constante global em uma variável e em uma variável do tipo estrutura.

### Declaração em Listas de Variáveis Globais

A variável Enable define qual bit deve ser acessado:

```
VAR_GLOBAL CONSTANT  
enable:int:=2;  
END_VAR
```

Exemplo 1, acesso a bit em uma variável inteira:

Declaração na POU:

```
VAR  
xxx:int;  
END_VAR
```

Bitaccess:

```
xxx.enable:=true; (* O terceiro bit na variável xxx será configurado para TRUE *)
```

Exemplo 2, acesso a bit em um componente de estrutura inteiro:

Declaração da estrutura stru1:

```
TYPE stru1 :
STRUCT
    bvar:BOOL;
    rvar:REAL;
    wvar:WORD;
    {bitaccess enable 42 'Start drive'}
END_STRUCT
END_TYPE
```

Declaração na POU:

```
VAR
    x:stru1;
END_VAR
```

Acesso a bit:

```
x.wvar.enable:=true;
```

Isto levará para TRUE o bit 42 na variável X. Uma vez que bvar tem 8 bits e rvar tem 32 bits, o acesso a bit será feito no segundo bit da variável wvar, que, como resultado, assumirá o valor 4.

## Endereços

NOTA: Alterações online podem alterar os conteúdos nos endereços. Considere isto ao usar ponteiros em endereços.

### Posição da Memória

A posição da memória pode ser acessada por qualquer tamanho suportado.

Por exemplo, o endereço %MD48 endereçaria os bytes 192, 193, 194 e 195 na área de posição da memória ( $48 * 4 = 192$ ). A posição do primeiro byte é zero. O exemplo abaixo mostra a posição da memória correspondente, dependendo do tamanho (X: bit, B: byte, W: word, D: dword), para endereçamento IEC.

Exemplo:

	Endereços			
<b>%MX</b>	96.0 - 96.7	96.8 - 192.15	97.0 - 97.7	97.8 - 97.15
<b>%MB</b>	192	193	194	195
<b>%MW</b>	96		97	
<b>%MD</b>		48		

**Tabela 4-13. Exemplos Posições de Memória**

Palavras, bytes e até mesmo bits podem ser acessados da mesma forma: o endereço %MX96.0 permite o acesso ao primeiro bit na word 96 (bits são geralmente salvos como words especiais).

Consulte **Endereço** para obter mais informações sobre endereçamento.

NOTA: Alterações online podem alterar os conteúdos nos endereços. Considere isto ao usar ponteiros em endereços.

### Endereço

Ao especificar um endereço, a posição de memória e o tamanho são indicados por uma sequência de caracteres especiais.

Sintaxe:

Endereço com bit:

%<Prefixo da área de memória><Tamanho do prefixo><número.número>

Endereço sem bit:

%<Prefixo da área de memória><Tamanho do prefixo><número>

Tipo	Descrição
I	Entrada (entradas físicas via driver de entrada, "sensores")
Q	Saída (saídas físicas via driver de saída, "atuadores").
M	Posição da memória

**Tabela 4-14. Prefixos da Área de Memória Suportados**

Tipo	Descrição
X	Bit único
B	Byte (8 bits)
W	Word (16 bits)
D	Double Word (32 bits)

**Tabela 4-15. Prefixos de Tamanho Suportados**

Exemplos:

Exemplo	Descrição
%QX7.5	Bit de saída 7.5
%IW215	Word de entrada 215
%QB7	Byte de saída 7
%MD48	Double Word na posição de memória 48
ivar AT %IW0 : WORD;	Exemplo de uma declaração de variável incluindo uma atribuição de endereço.

**Tabela 4-16. Exemplos de Endereçamento**

Para atribuir um endereço válido em uma aplicação, primeiramente é necessário conhecer a posição adequada da memória na imagem do processo, ou seja, a área de memória a ser usada: Entrada (I), Saída (Q) ou Memória (M), conforme mencionado acima. A seguir, especifique o tamanho desejado: bit, byte, word e dword (veja acima: X, B, W, D).

A configuração atual do dispositivo e as definições (estrutura de hardware, descrição do dispositivo, configurações de E/S) têm um papel decisivo no endereçamento. Considere especialmente as diferenças na interpretação dos endereços entre dispositivos usando Modo de Endereçamento de Byte ou aquelas usando o Modo de Endereçamento IEC orientado à Word. Assim, dependendo do tamanho e do modo de endereçamento, células diferentes de memória podem ser endereçadas pela mesma definição de endereço.

Na Tabela 4-17, veja uma comparação entre o endereçamento de byte e o endereçamento IEC orientado à word para bits, bytes, palavras e palavras duplas. Visualize ainda as áreas de sobreposição de memória no caso do modo de endereçamento de byte.

Em relação à notação, considere que, para endereços de bit, o modo de endereçamento é sempre orientado à word, ou seja, a posição antes do ponto corresponde ao número da word e a posição após o nome representa o número do bit.

Obs.: n = número do byte.

DWords/Words				Bytes	X (bits)					
Endereçamento a Byte		Endereçamento orientado a Words IEC			Endereçamento a Byte			Endereçamento orientado a Words IEC		
D0	W0	D0	W0	B0	X0.7	...	X0.0	X0.7	...	X0.0
D1	W1			B1	X1.7	...	X1.0	X1.15	...	X0.8
	W2		W1	B2	...			X1.7	...	X1.0
	W3			B3				X1.15	...	X1.8
	W4	D1	W2	B4						
	...			B5						
			W3	B6						
				B7						
		D2		B8						
		...		...						
		...		...						
		...		...						
D(n-3)		D(n/4)	...							
	W(n-1)		W(n/2)							
				Bn	Xn.7	...	Xn.0	X(n/2).15	...	X(n/2).8

**Tabela 4-17. Comparação Endereçamento Byte X Word (D,W,B,X)**

Sobreposição dos intervalos de memória em caso de modo de endereçamento de byte:

D0 contém B0 - B3. W0 contém B0 e B1. W1 contém B1 e B2. W2 contém B2 e B3. Para acessar a área fora da sobreposição, W1, D1, D2 e D3 não devem ser usados para endereçamento.

**NOTAS:**

- Valores booleanos serão distribuídos por byte, se nenhum endereço de bit único for explicitamente especificado. Exemplo: uma alteração no valor de varbool1 AT %QW0 afeta o intervalo de QX0.0 a QX0.7.
- Alterações online podem alterar os conteúdos nos endereços. Considere isto ao usar ponteiros em endereços.

## Funções

Em ST, uma chamada de função pode também funcionar como um operando.

Exemplo:

```
Result := Fct(7) + 3;
```

### TIME()-Function

Esta função retorna o tempo (em milissegundos) decorrido desde a inicialização do sistema.

O tipo de dado é TIME.

Exemplo em IL:

```
1 | TIME
2 | ST      tempo
```

Exemplo em ST:

```
tempo := TIME ();
```

## 5. Editores das Linguagens de Programação

### Editor CFC

O editor CFC está disponível para objetos de programação na linguagem Gráfico Funcional Contínuo (CFC), que é uma extensão das linguagens de programação IEC 61131-3. Selecione a linguagem ao adicionar um novo objeto POU via comando *Acrescentar Objeto* no projeto.

O editor CFC é um editor gráfico.

O editor estará disponível na parte inferior da janela que se abre quando o objeto POU CFC é aberto. Na sua parte superior, a janela mostra o *Editor de Declaração*.

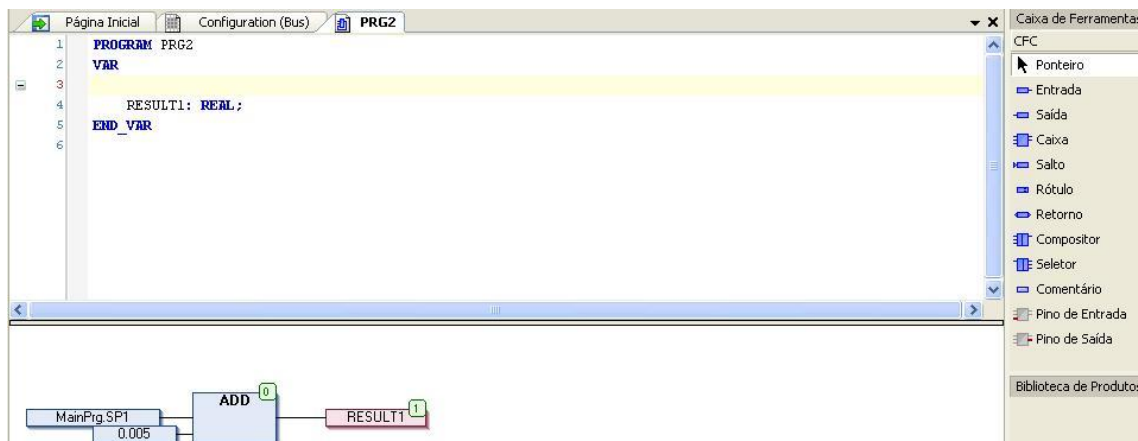




Figura 5-1. Editor CFC

O editor CFC, ao contrário dos editores de rede, permite o livre posicionamento dos elementos, possibilitando a inserção de caminhos de realimentação. A sequência de processamento é determinada por uma lista contendo todos os elementos atualmente inseridos, a qual pode ser modificada.

Os seguintes elementos estão disponíveis na *Caixa de Ferramentas* para inserção: caixa (operadores, funções, blocos funcionais e programas), entrada, saída, comentário, rótulo, salto, compositor, seletor, retorno e pinos de entrada/saída.

As linhas de entrada e saída dos elementos podem ser conectadas através do mouse (arraste da conexão). O curso desta linha será criado automaticamente, seguindo o menor caminho possível. As linhas de conexão serão automaticamente ajustadas assim que os elementos forem movidos. Veja também: **Inserir e Organizar Elementos** neste capítulo.

A dimensão da janela do editor pode ser alterada através do zoom: use o botão  no canto inferior direito da janela e escolha uma das opções disponíveis. Também é possível selecionar o item  e abrir um diálogo para digitar o fator desejado.

Os comandos para trabalhar no Editor CFC podem ser chamados através do menu de contexto ou do menu *CFC* (disponível assim que este editor for aberto).

### Linguagem Gráfico Funcional Contínuo - CFC

O Gráfico Funcional Contínuo, é uma extensão da norma IEC 61131-3, e consiste em uma linguagem de programação gráfica baseada no Diagrama de Blocos Funcionais. Entretanto, diferentemente do FBD, não são utilizadas redes, mas sim o livre posicionamento de elementos gráficos, permitindo, por exemplo, laços de realimentação.

Para criar objetos de programação CFC no MasterTool IEC XE, consulte o **Editor CFC**.

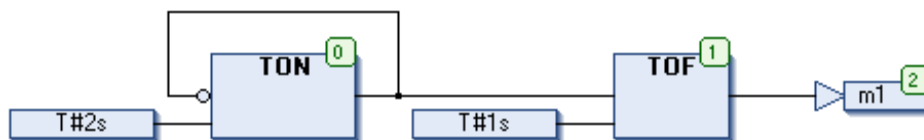


Figura 5-2. Exemplo de uma Rede CFC

### Posições do Cursor no CFC

Em um programa CFC, as posições possíveis do cursor são indicadas, por padrão, por um sombreado cinza quando o mesmo é posicionado sobre os elementos.

Assim que o usuário clicar em uma área sombreada (antes de soltar o mouse), a mesma mudará para a cor vermelha. Ao soltar o botão, o cursor estará posicionado nesta área e o respectivo elemento ou texto selecionado serão exibidos em vermelho.

Há três categorias de posições de cursor. Veja, na Figura 5-3, Figura 5-4 e Figura 5-5, as possibilidades indicadas por uma área sombreada em cinza.

- Se o cursor estiver posicionado em um texto, este será exibido em azul e poderá ser editado. O botão [...] abre o *Assistente de Entrada*. O texto “???” deve ser substituído por um identificador válido:



Figura 5-3. Possíveis Posições do Cursor e Dois Exemplos de Textos Selecionados

- Se o cursor estiver posicionado no corpo de um elemento (caixa, entrada, saída, salto, rótulo, retorno e comentário), este será exibido em vermelho e poderá ser movido com o mouse:

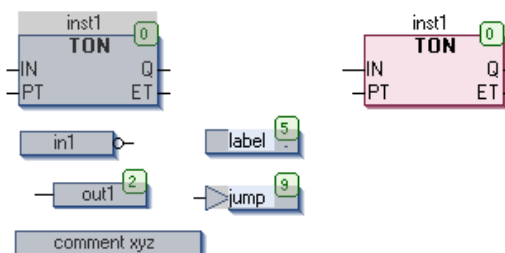


Figura 5-4. Possíveis Posições do Cursor e Exemplo de um Corpo Selecionado

- Se o cursor estiver posicionado em uma conexão de entrada ou saída de um elemento, este será exibido em vermelho e pode ser negado ou submetido a um set/reset:

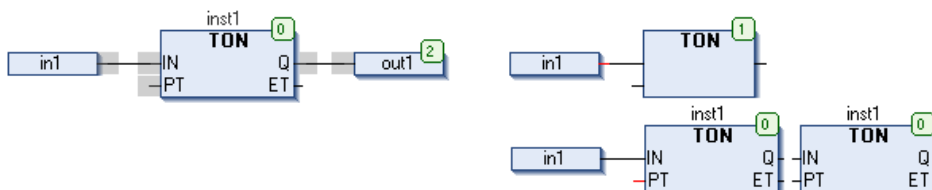


Figura 5-5. Possíveis Posições do Cursor e Exemplos de Posições de Entrada e Saída Selecionadas

## Elementos CFC / Caixa de Ferramentas

Os elementos gráficos disponíveis para programação na janela do editor CFC são fornecidos por uma *Caixa de Ferramentas*. Ela pode ser aberta em uma janela de visualização através do comando correspondente no menu *Visualizar*.



**Figura 5-6. Caixa de Ferramentas CFC, Padrão**

O elemento desejado pode ser selecionado na *Caixa de Ferramentas* e inserido na janela do editor arrastando e soltando.

Além dos elementos de programação, no topo da lista da *Caixa de Ferramentas* encontra-se o item **Ponteiro**. Enquanto este item estiver selecionado, o cursor transforma-se em uma seta e, com ela, é possível selecionar os elementos na janela do editor para posicioná-los e editá-los.

Elementos:

Símbolo	Pino	Representação	Significado
	Entrada		O texto "???" pode ser selecionado e substituído por uma constante. O diálogo Assistente de Entrada serve para selecionar um identificador válido para a entrada.
	Saída		O texto "???" pode ser selecionado e substituído por uma constante. O diálogo Assistente de Entrada serve para selecionar um identificador válido para a saída.
	Caixa		Uma caixa pode ser usada para representar operadores, funções, blocos funcionais e programas. O texto "???" pode ser selecionado e substituído pelos objetos acima citados (usando o Assistente de Entrada). Caso um bloco funcional seja inserido, outro texto "???" será exibido acima da caixa e deve ser substituído pelo nome da instância do bloco funcional. Caso uma caixa existente seja substituída por outra (alterando o nome) e a nova tenha um número diferente de pinos de entradas ou saída, estes pinos sofrerão uma adaptação. Caso alguns pinos devam ser excluídos, o inferior será removido primeiro.
	Salto		O salto é usado para indicar em qual posição a execução do programa deve continuar. A posição é definida por um rótulo. Substitua o texto "???" pelo nome do rótulo.
	Rótulo		Um rótulo marca a posição para qual o programa




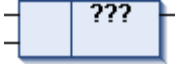

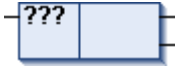



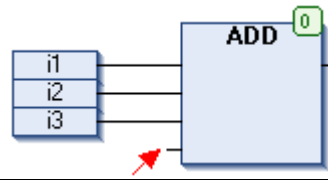

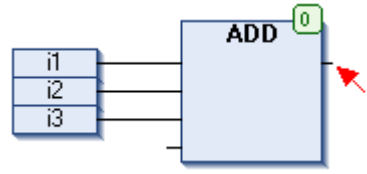
	<b>Retorno</b>		<p>saltará (veja acima Salto).</p> <p>No modo online, um rótulo de RETORNO é automaticamente inserido para marcar o final da POU. Observe que, no modo online, o elemento RETURN é automaticamente inserido na primeira coluna e após o último elemento no editor. No passo a passo, ele automaticamente salta para esta posição antes que a execução saia da POU.</p>
	<b>Compositor</b>		<p>Um compositor é usado para tratar uma entrada de caixa do tipo estrutura. O compositor exibe os componentes da estrutura e os disponibiliza para o programador no CFC. Para tal, nomeie o compositor da mesma forma que a referida estrutura (substituindo "???" pelo nome) e conecte-o à caixa em vez de usar uma Entrada.</p>
	<b>Seletores</b>		<p>Um seletor, ao contrário do compositor, é usado para tratar uma saída de caixa do tipo estrutura. O seletor exibe os componentes da estrutura e os disponibiliza para o programador no CFC. Para tal, nomeie o seletor da mesma forma que a referida estrutura (substituindo "???" pelo nome) e conecte-o à caixa em vez de usar uma Saída.</p>
	<b>Comentário</b>		<p>Use este elemento para incluir comentários no gráfico. Selecione o texto do espaço reservado e substitua-o pelo texto desejado. Uma nova linha pode ser incluída no comentário pressionando &lt;Ctrl&gt; + &lt;Enter&gt;.</p>
	<b>Pino de Entrada</b>		<p>Dependendo do tipo de caixa, uma entrada adicional pode ser adicionada. Para tanto, selecione a caixa na rede CFC e desenhe nela o pino de entrada.</p>
	<b>Pino de Saída</b>		<p>Dependendo do tipo de caixa, uma saída adicional pode ser adicionada. Para tanto, selecione a caixa na rede CFC e desenhe nela o pino de saída.</p>

Tabela 5-1. Elementos na Janela do Editor

Exemplo do elemento *Compositor*:

Suponha um programa CFC `cfc_prog`, lido com uma instância do bloco funcional `fubblo1`, que apresenta a variável de entrada `struvar` do tipo estrutura. Usando o *Compositor*, os componentes da estrutura podem ser acessados.



Definição de estrutura stru1:

```
TYPE stru1 :
STRUCT
  ivar:INT;
  strvar:STRING:='hallo';
END_STRUCT
END_TYPE
```

Bloco funcional fublo1, declaração e implementação:

```
FUNCTION_BLOCK fublo1
VAR_INPUT
  struvar:STRU1;
END_VAR
VAR_OUTPUT
  fbout_i:INT;
  fbout_str:STRING;
END_VAR
VAR
  fbvar:STRING:='world';
END_VAR
fbout_i:=struvar.ivar+2;
fbout_str:=CONCAT (struvar.strvar, fbvar);
```

Programa cfc\_prog, declaração e implementação:

```
PROGRAM cfc_prog
VAR
  intvar: INT;
  stringvar: STRING;
  fbinst: fublo1;
  erg1: INT;
  erg2: STRING;
END_VAR
```

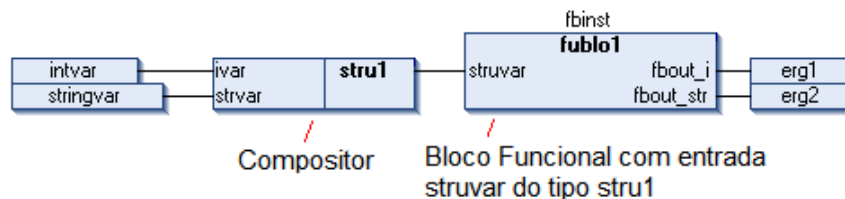


Figura 5-7. Exemplo Compositor

Exemplo do elemento *Selector*: suponha um programa CFC cfc\_prog, lido com uma instância do bloco funcional fublo2, que apresenta a variável de saída fbout do tipo estrutura stru1. Usando o *Selector*, os componentes da estrutura podem ser acessados.

Definição da estrutura stru1:

```
TYPE stru1 :
STRUCT
  ivar:INT;
  strvar:STRING:='hallo';
END_STRUCT
END_TYPE
```

Bloco funcional fublo2, declaração e implementação:

```
FUNCTION_BLOCK fublo2
VAR_INPUT CONSTANT
fbin1:INT;
fbin2:DWORD:=24354333;
fbin3:STRING:='hallo';
END_VAR
VAR_INPUT
  fbin : INT;
END_VAR
VAR_OUTPUT
fbout : stru1;
fbout2:DWORD;
END_VAR
VAR
fbvar:INT;
  fbvar2:STRING;
END_VAR
```

Programa cfc\_prog, declaração e implementação:

```
VAR
intvar: INT;
stringvar: STRING;
fbinst: fublo1;
erg1: INT;
erg2: STRING;
fbinst2: fublo2;
END_VAR
```

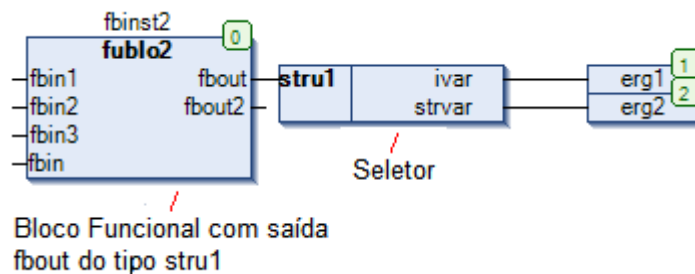


Figura 5-8. Exemplo Seletor

## Inserir e Organizar Elementos

Os elementos disponíveis para programação no editor CFC são fornecidos em uma *Caixa de Ferramentas* que, está disponível em uma janela assim que o editor CFC for aberto.

O diálogo *Opções do Editor CFC* define as configurações gerais para se trabalhar neste editor.

### Inserindo

Para inserir um elemento, selecione-o na *Caixa de Ferramentas* que com um clique do mouse, mantenha o botão pressionado e posicione-o no local desejado na janela do editor. Durante o posicionamento, o cursor será exibido como uma seta com um retângulo e um sinal de mais. Ao soltar o botão, o elemento será inserido.

### Selecionando

Para selecionar um elemento inserido visando outras ações (editar ou reorganizar), verifique as possíveis posições do cursor para corpos de elemento, entradas/saídas e texto. Com um clique do mouse no corpo dos elementos, os mesmos são selecionados e exibidos na cor vermelha. Mantendo

pressionada a tecla <SHIFT> outros elementos podem ser selecionados ao mesmo tempo. É possível, também, pressionar o botão esquerdo do mouse e desenhar um retângulo pontilhado em volta dos elementos a serem selecionados. Ao soltar o botão, a seleção será indicada. Através do comando *Selecionar Todos*, disponível no menu de contexto, todos os elementos são selecionados de uma vez só.

Usando as teclas de seta, pode-se alterar a seleção entre os elementos selecionáveis. A sequência depende da ordem de execução dos elementos, indicada por seus números.

Quando um pino de entrada for selecionado e as teclas <CTRL>+<SETA ESQUERDA> forem pressionadas simultaneamente, a saída correspondente será selecionada também. Quando um pino de saída for selecionado e as teclas <CTRL>+<SETA ESQUERDA> forem pressionadas simultaneamente, as saídas correspondentes serão selecionadas também.

### *Substituindo Caixas*

Para substituir um elemento caixa, substitua o identificador atualmente inserido pelo identificador do novo elemento desejado. Se aplicável, o número dos pinos de entrada e saída serão adaptados devido à definição das POUs e, assim, algumas atribuições existentes podem ser removidas.

### *Movendo*

Para mover um elemento, selecione o seu corpo com um clique do mouse e arraste-o até a posição desejada (mantendo o botão pressionado). Então, solte o botão e posicione o elemento. Os comandos *Recortar* e *Colar* também podem ser utilizados para esta ação.

### *Conectando*

As conexões entre as entradas e saídas de elementos podem ser desenhadas com o mouse. A menor conexão possível será criada considerando os outros elementos e conexões. A cor verde clara no curso das linhas de conexão indica que os elementos estão posicionados um sobre os outros.

### *Copiando*

Para copiar um elemento, selecione-o e use os comandos *Copiar* e *Colar*.

### *Editando*

Após inserir um elemento, a parte do texto é representado por “???”. Para substituir este sinais pelo texto desejado (nome da POU, do rótulo, da instância, comentário...) selecione o texto com um clique do mouse para entrar em um campo de edição. O botão também estará disponível para abrir o *Assistente de Entrada*.

### *Excluindo*

Um elemento pode ser excluído através do comando *Excluir*, disponível no menu de contexto ou através da tecla <DEL>.

### *Ordem de Execução, Números de Elemento*

A sequência na qual os elementos em uma rede CFC são executados no modo online é indicada por números no canto superior direito dos elementos caixa, saída, salto e retorno. O processamento inicia no elemento de número mais baixo (0). A ordem de execução pode ser modificada através dos comandos disponíveis no submenu *Ordem de Execução* no menu CFC.

Ao adicionar um elemento, o número será definido automaticamente de acordo com a sequência topológica (da esquerda para a direita e de cima para baixo). Os novos elementos recebem o número do seu sucessor se a sequência já foi alterada e todos os números mais altos são aumentados em um.

O número de um elemento permanece constante quando este é movido.

A sequência influencia o resultado e, em alguns casos, deve ser alterada.

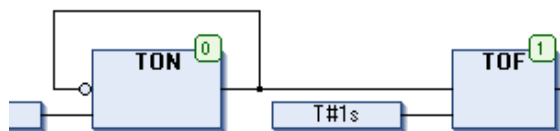


Figura 5-9. Exemplo de Números de Elemento

### Alterando o Tamanho da Planilha de Trabalho

Para obter mais espaço no entorno de um gráfico CFC na janela de um editor, o tamanho da área de trabalho (planilha de trabalho) pode ser alterado. Isto pode ser feito selecionando e arrastando todos os elementos com o mouse, ou através dos comandos *Recortar* e *Colar* (veja acima o item **Movendo**).

Para poupar tempo (no caso de gráficos maiores), outra opção é utilizar o diálogo de configurações de dimensões especiais. Consulte o item **Editar Planilha de Trabalho** no Manual de Utilização MasterTool IEC XE – MU299048.

### Editor CFC no Modo Online

No modo online, o editor CFC fornece visualizações para monitoração, escrita e forçamento das variáveis e expressões no controlador. A funcionalidade *Depuração* (breakpoints, passo a passo, etc.) está disponível.

Para obter informações sobre como abrir objetos no modo online, consulte **Interface do Usuário no Modo Online** no Manual de Utilização MasterTool IEC XE – MU299048.

A janela do editor em um objeto CFC também inclui o *Editor de Declaração* na parte superior. Para obter informações sobre o editor de declaração no modo online, consulte **Editor de Declaração no Modo Online** no Manual de Utilização MasterTool IEC XE – MU299048.

### Monitoração

Os valores atuais são exibidos em pequenas janelas de monitoração em cada variável (monitoração em linha).

Device.Application.PRG2				
Expressão	Tipo	Valor	Valor Preparado	Comentário
RESULT1	REAL	60.425		

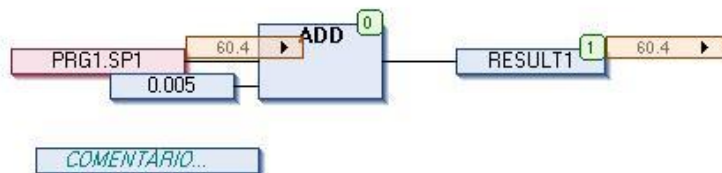
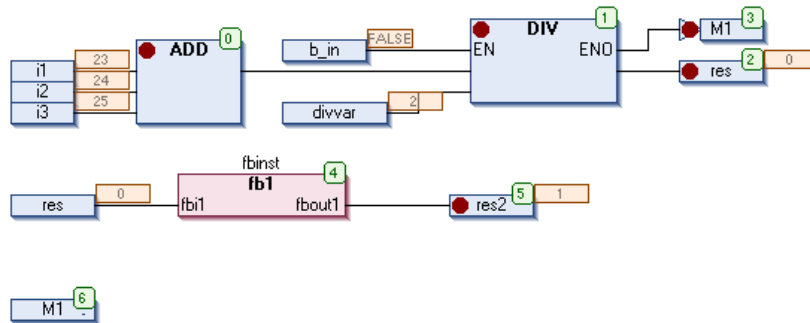


Figura 5-10. Visualização Online de um Objeto de Programa MAINPRG

Na visualização online de uma POU do tipo bloco funcional, os valores não serão visualizados nas janelas de monitoração na parte de implementação, porém o “<Valor da expressão>” e os campos de monitoração em linha exibirão três pontos de interrogação.

### Posições de Breakpoint no Editor CFC

As posições de breakpoint possíveis, basicamente, são aquelas da POU na qual os valores das variáveis podem mudar, ou na qual o programa se ramifica ou outra POU é chamada. As posições possíveis são mostradas na Figura 5-11.



**Figura 5-11. Posições de Breakpoint no Editor CFC**

NOTA: Nos métodos, um breakpoint será configurado automaticamente em todos os métodos em que puder ser chamado. Se um método for chamado via um ponteiro em um bloco funcional, os breakpoints serão configurados no método do bloco funcional e em todos os blocos funcionais derivados que o subscrevem.

### Editor SFC

O editor SFC está disponível para objetos de programação na linguagem Sequenciamento Gráfico de Funções (SFC) da norma IEC 61131-3. A linguagem deve ser escolhida no momento em que uma nova POU é adicionada ao projeto via comando *Acrrescentar Objeto*.

O editor SFC é um editor gráfico. Configurações gerais referentes ao comportamento e exibição são feitas no diálogo *Opções do Editor SFC*.

O editor SFC está disponível na parte inferior da janela que se abre ao editar uma POU SFC. Na sua parte superior, aparece o *Editor de Declaração*.

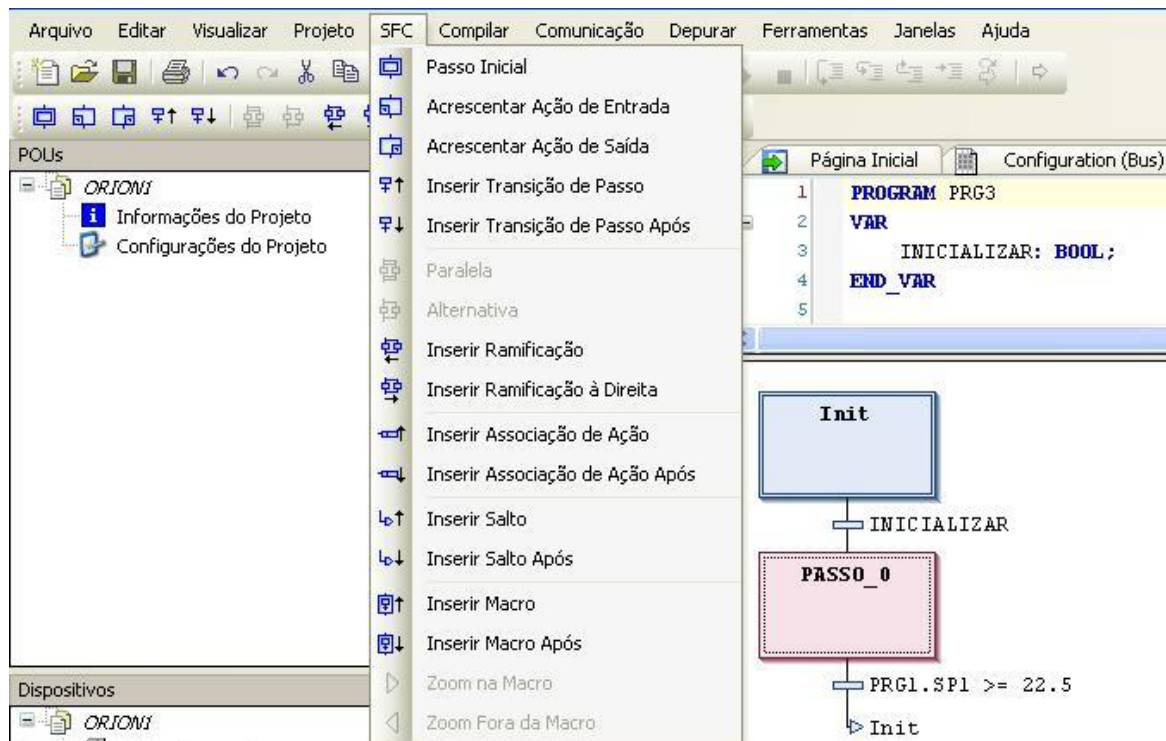


Figura 5-12. Editor SFC

Os elementos usados em um diagrama SFC encontram-se disponibilizados assim que o editor SFC for ativado. Eles são organizados em uma sequência de passos conectados por transições. Veja também: **Trabalhando no Editor SFC** neste capítulo.

As propriedades dos passos podem ser editadas na caixa *Propriedades*. Neste campo são definidos os tempos de ativação (mínimo e máximo) para cada passo.

Variáveis implícitas podem ser acessadas para controlar o processamento de um SFC (por exemplo, status de passo, análise de timeout, reset e outros).

Os comandos para trabalhar no editor SFC podem ser chamados a partir do menu de contexto ou do menu *SFC*.

No editor SFC:

- A edição é facilitada pelo fato de que cada elemento específico pode ser selecionado e organizado individualmente. Durante a edição, não necessariamente deve haver correspondência de sintaxe do SFC. Os erros de sintaxe não serão verificados até a execução do comando *Gerar Código*.
- Há somente um tipo de passo que combina os estilos tradicionais (passos IEC e passos não IEC). As ações sempre devem ser fornecidas como POU's e sempre são atribuídas via propriedades de elemento de passo.
- Macros podem ser usadas para fins de estruturação.

### Sequenciamento Gráfico de Funções - SFC

O Sequenciamento Gráfico de Funções (SFC) é uma linguagem gráfica orientada que permite descrever a ordem cronológica de ações determinadas em um programa. Estas ações estão disponíveis como objetos de programação separados, escritos em qualquer linguagem de programação disponível. Em um SFC, elas são atribuídas a elementos de passo e a sequência de processamento é controlada por elementos de transição. Para obter uma descrição detalhada sobre como os passos serão processados no modo online, consulte o item **Sequência de Processamento no SFC** neste capítulo.

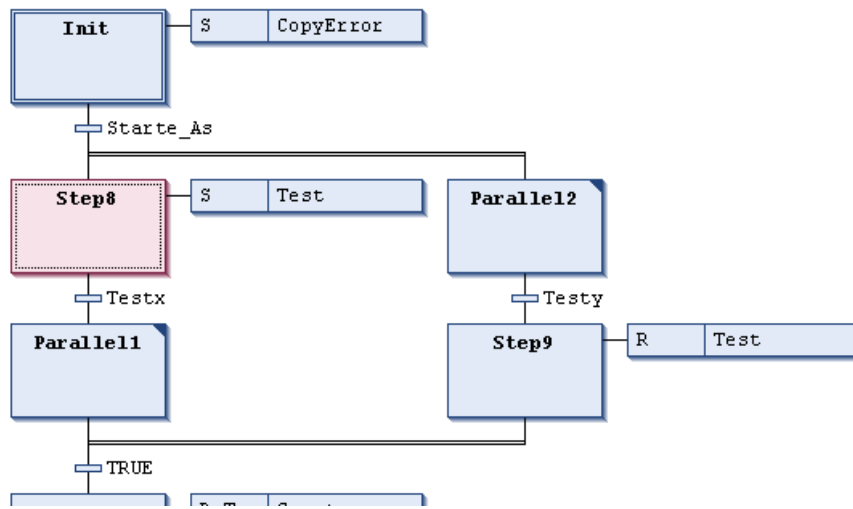


Figura 5-13. Exemplo para uma Sequência de Passos em um Módulo SFC

### Posições do Cursor no SFC

As posições possíveis do cursor em um diagrama SFC no editor SFC são indicadas, por padrão, por um sombreado cinza ao passar o cursor sobre os elementos.

Há duas categorias de posições de cursor: textos e corpos de elemento. Veja as posições possíveis indicadas pela área sombreada nos próximos capítulos.

#### Textos

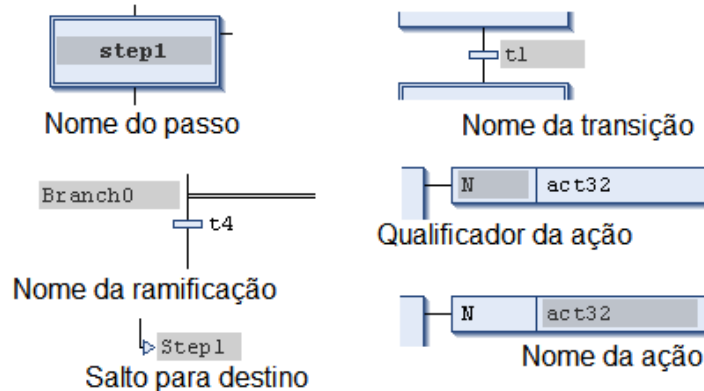


Figura 5-14. Posições Possíveis do Cursor, Textos

Ao clicar em uma posição de texto do cursor o texto poderá ser editado.

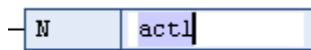


Figura 5-15. Seleção do Nome da Ação para Edição

## Corpos de Elemento

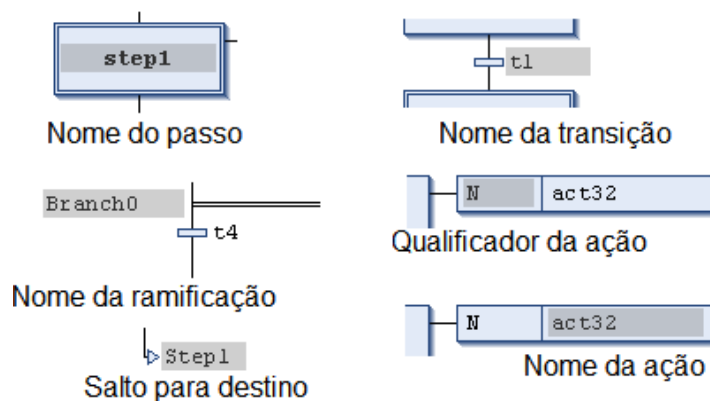


Figura 5-16. Posições Possíveis do Cursor, Corpos de Elemento

Ao clicar em uma área sombreada, o elemento será selecionado. Ele adquire uma moldura pontilhada e é exibido em sombreado vermelho (para selecionar vários elementos, veja abaixo **Trabalhando no Editor SFC**).

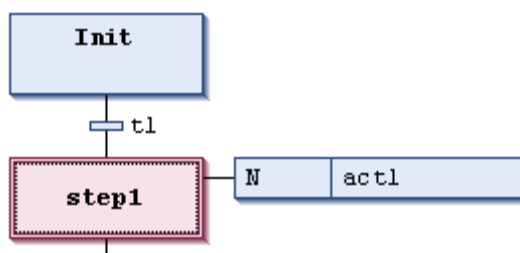


Figura 5-17. Elemento de Passo Selecionado

## Trabalhando no Editor SFC

A princípio, uma nova POU SFC contém um passo inicial e uma transição subsequente. Para saber como adicionar outros elementos, como organizá-los e editá-los, veja as informações a seguir.

Para verificar as posições possíveis do cursor, veja o item acima.

É possível navegar, isto é, pular para o próximo elemento ou para o elemento anterior no gráfico, usando as teclas de seta.

## Inserir Elementos

Os elementos SFC específicos podem ser inseridos através dos respectivos comandos (por padrão, disponíveis no menu *SFC*). Consulte o item **Posições do Cursor no SFC** para obter um detalhamento. Um clique duplo em um passo, transição ou ação já inseridos, que ainda não tenham sido referenciados no projeto, abrirá um diálogo para atribuição.

## Selecionar Elementos

Um elemento e um campo de texto podem ser selecionados por um clique do mouse em uma posição possível do cursor. A seleção pode ser estendida aos elementos adjacentes usando as teclas de seta. O elemento será alterado para a cor vermelha. Consulte exemplos no item **Posições do Cursor no SFC**.

Passos e transições podem ser selecionados, movidos (recortar, copiar, colar) ou excluídos separadamente.

Para selecionar vários elementos, siga as instruções abaixo.

- Mantenha a tecla <SHIFT> pressionada e, em seguida, clique nos elementos a serem selecionados.



- Pressione o botão esquerdo do mouse e desenhe um retângulo (linha pontilhada) em volta dos elementos a serem selecionados.
- Use o comando *Selecionar Todos* (menu *Editar*).

### *Editar Textos*

Com um clique do mouse em um texto, abre-se um campo de edição. Se uma área de texto foi selecionada via teclas de seta, o campo de edição deve ser aberto explicitamente via <ESPAÇO>.

### *Editar Ações Associadas*

Um clique duplo em um passo (entrada, ativo ou saída) ou ação de transição associada abre a ação associada no editor correspondente. Por exemplo, dê um duplo clique no elemento transição ou no triângulo indicando uma ação de saída em um elemento de passo.

### *Recortar, Copiar e Colar Elementos*

Selecione o(s) elemento(s) e use o comando *Recortar*, *Copiar* ou *Colar* (menu *Editar*) ou as teclas correspondentes.

Observe o seguinte comportamento:

- Ao copiar um ou vários elementos, o conteúdo da área de transferência será inserido antes da posição atualmente selecionada. Se não houver nada selecionado no momento, os elementos serão anexados ao final do gráfico atualmente carregado.
- Se uma ramificação for colada e o elemento atualmente selecionado também for uma ramificação, as ramificações copiadas serão inseridas à esquerda daquelas já existentes.
- Ao colar uma ação (lista) em um passo atualmente selecionado, as ações serão adicionadas no início da lista de ações do passo ou então, uma lista de ação para o passo será criada.
- Elementos incompatíveis ao recortar/copiar: se uma ação associada (lista) e um elemento adicional, que não é o passo ao qual a ação (lista) pertence, forem selecionados, uma caixa de mensagem aparecerá: “A seleção atual contém elementos incompatíveis. Os dados não serão levados para a área de transferência.”. A seleção não será armazenada e não será possível colá-los ou copiá-los em outro lugar.
- Elementos incompatíveis ao colar: se o elemento atualmente selecionado não for um passo ou outra associação, ao tentar colar uma ação (lista), uma caixa de erro aparecerá: “O conteúdo atual da área de transferência não pode ser colado na seleção atual.”. A mesma mensagem aparecerá ao tentar colar um elemento como um passo, ramificação ou transição quando uma ação associada (lista) estiver selecionada no momento.

### *Excluir Elementos*

Selecione os elementos e use o comando *Excluir* ou a tecla <DEL>. Considere:

- Excluir um passo exclui também a lista de ações associada.
- Excluir o passo inicial automaticamente define que o passo seguinte será o inicial. A opção *Passo Inicial* será ativada nas propriedades deste passo.
- Excluir a linha horizontal que precede uma área ramificada exclui todas as ramificações.
- Excluir todos os elementos específicos de uma ramificação exclui a ramificação.

### **Propriedades do Elemento SFC**

As propriedades de um elemento SFC podem ser visualizadas e editadas na janela aberta via comando *Propriedades do Elemento* (no menu *Visualizar*).

As propriedades que serão exibidas dependem do elemento atualmente selecionado. As propriedades são dispostas em grupos, os quais são abertos ou fechados através dos sinais de mais e menos, respectivamente.

Na guia *Visualizar* das opções do *Editor SFC* (menu *Ferramentas, Opções*), é definido quais propriedades devem ser exibidas ao lado do elemento no gráfico SFC.

Veja a seguir todas as propriedades possíveis.

Comum:

Propriedade	Descrição
<b>Nome</b>	Nome do elemento, a princípio: <elemento><número de execução>. Exemplo: Step0 (nome do passo), Step1, branch0 (nome da ramificação), etc.
<b>Comentário</b>	Comentário do elemento (string de texto), por exemplo "Zerar o contador". Quebras de linha podem ser inseridas via <Ctrl>+<Enter>.
<b>Símbolo</b>	Para cada item SFC, uma memória é criada implicitamente com o mesmo nome do elemento. Esta opção é usada para definir se esta memória deve ser exportada para a configuração de símbolos e como a mesma deve ser acessada no CP. Execute um duplo clique no campo valor, selecione-o e use a tecla de espaço para abrir a lista de seleção para escolher uma das opções de acesso. Nenhuma: o símbolo será exportado para a configuração de símbolos, mas não estará acessível no CP. Leitura: o símbolo será exportado para a configuração de símbolos e poderá ser lido no CP. Escrita: o símbolo será exportado para a configuração de símbolos e será escrito no CP. Leitura/escrita: combinação de leitura e escrita. A princípio, nada é inserido neste campo, ou seja, o símbolo não é exportado para a configuração de símbolos.

**Tabela 5-2. Descrição das Propriedades Comuns**

Específico:

Propriedade	Descrição
<b>Passo inicial</b>	Esta ação sempre está ativada nas propriedades do passo inicial atual. A princípio, ele está ativado para o primeiro passo em um SFC e desativado para outros passos. Ao ativar esta opção em outro passo, <b>desative-o</b> no passo anterior para evitar um erro de compilação.
<b>Tempos:</b>	Atente para a possibilidade de detectar timeouts em passos utilizando a memória SFCErrror.
<b>Ativação mínima</b>	Tempo de processamento mínimo do passo. Valores permitidos: tempo de acordo com a sintaxe IEC (por exemplo t#8s) ou variável TIME; o padrão é t#0s.
<b>Ativação máxima</b>	Tempo de processamento máximo do passo. Valores permitidos: tempo de acordo com a sintaxe IEC (por exemplo t#8s) ou variável TIME; o padrão é t#0s.
<b>Ações</b>	Define as ações a serem executadas quando o passo estiver ativo. Veja a descrição da sequência de processamento.
<b>Entrada de passo</b>	Esta ação será executada após o passo ter sido ativado.
<b>Passo ativo</b>	Esta ação será executada quando o passo estiver ativo e as eventuais ações de entrada já tiverem sido processadas.
<b>Saída do passo</b>	Esta ação será executada no ciclo subsequente após um passo ter sido desativado.

**Tabela 5-3. Descrição das Propriedades Específicas**

NOTA: Considere a possibilidade de obter informações sobre o status do passo/ação, timeouts, etc. através das variáveis implícitas e memórias SFC apropriadas.

### Elementos SFC / Caixa de Ferramentas

Os elementos gráficos usados para a programação na janela do editor SFC podem ser inseridos através dos comandos de inserção (menu *SFC*).

Consulte também o item **Trabalhando no Editor SFC**.

Os seguintes elementos estão disponíveis e são descritos a seguir:

- Passo
- Transição
- Ação
- Ramificação (Alternativa)
- Ramificação (Paralela)
- Salto
- Macro

### Transição de Passo

Símbolo: 

Um passo é representado por uma caixa (contendo o seu nome) conectada às transições precedente e subsequente através de uma linha.

O nome do passo pode ser editado na linha.

A moldura da caixa do passo inicial apresenta linha dupla.

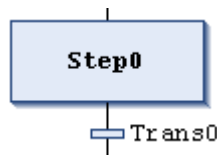
Todo o passo - através do comando Passo Inicial ou da respectiva propriedade do passo - pode ser transformado em um passo inicial. Em outras palavras, pode ser convertido no passo que será executado primeiro quando a POU for chamada.

As ações a serem executadas quando o passo está ativo (processado) devem estar associadas (veja abaixo, **Ação**).

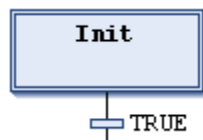
Passos e transições são basicamente inseridos juntos via comando *Inserir Transição de Passo Após*.

#### NOTAS:

- Somente os tipos de passo IEC estão previstos.
- Nomes de passo devem ser exclusivos no escopo de uma POU principal. Isto deve ser observado especialmente ao usar ações programadas em SFC.



**Figura 5-18. Passo e Subsequente Transição**



**Figura 5-19. Passo Inicial e Subsequente Transição**

### Transição

Uma transição é representada por um pequeno retângulo conectado ao antecessor e ao sucessor dos passos por uma linha. Ela fornece a condição na qual o passo seguinte se tornará ativo (assim que a condição for TRUE).

Por padrão, automaticamente é inserida uma transição “trans<n>”, onde n é o número de execução.

O nome padrão deve ser modificado por um nome válido, seja ele:

- O nome de um objeto de transição (📄) disponível na árvore das POUs (o que permite o uso de várias transições. Veja, por exemplo “t1” na coluna esquerda)
- Uma expressão condicional válida

NOTA: Considere que transições do tipo objeto ou propriedade são indicadas por um pequeno triângulo no canto superior direito do retângulo.

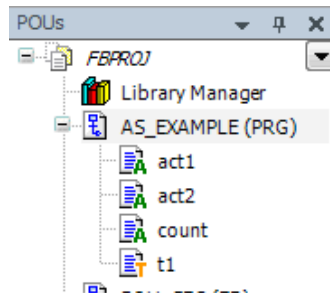


Figura 5-20. Transição na Árvore das POUs

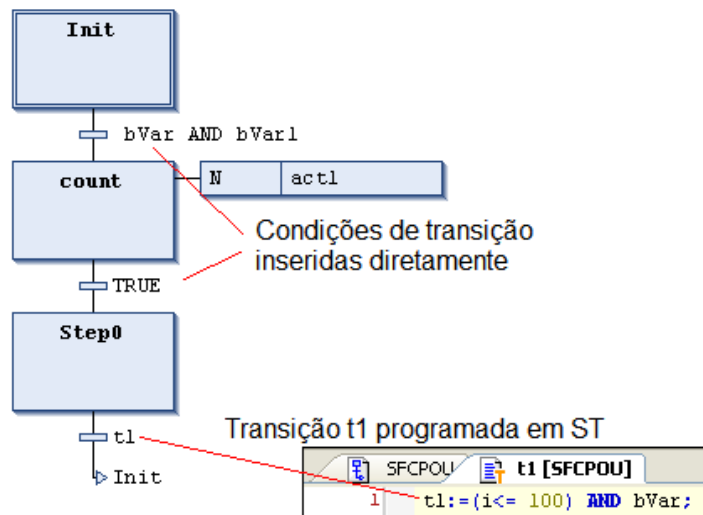


Figura 5-21. Exemplos de Transição

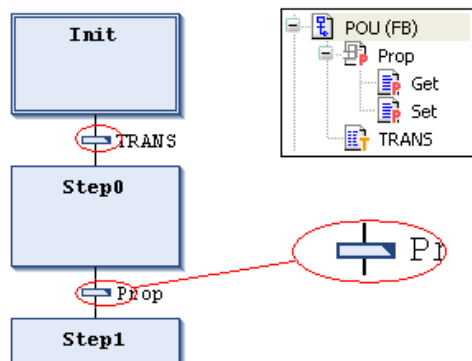


Figura 5-22. Transição ou Propriedade Indicados por um Triângulo

Uma condição de transição deve ter os valores TRUE ou FALSE. Assim sendo, ela consiste em uma variável booleana, um endereço booleano, uma constante booleana ou uma série de instruções com um resultado booleano. No entanto, uma transição não pode conter programas, blocos funcionais ou atribuições.

Uma condição de transição é tratada como uma chamada de método. Ela será inserida de acordo com a sintaxe:

```
<Nome da transição>:=<condição da condição>; (* por exemplo "trans1:=a=100" *)
```


Ou:

```
<Condição de transição>; (* por exemplo "a=100" *)
```


Veja também t1 na coluna esquerda.

No modo online, o passo subsequente apenas pode tornar-se ativo se a transição precedente tornar-se TRUE.

## Ação

Símbolo: 

Uma ação pode conter uma série de instruções escritas em uma das linguagens de programação válidas. Ela é atribuída a um passo e, no modo online, será processada de acordo com a sequência de processamento definida.

Cada ação a ser usada em passos SFC deve estar disponível como uma POU válida na POU SFC e no projeto ()

O comando *A acrescentar Objeto* está disponível para adicionar uma ação (POU) em uma POU SFC.

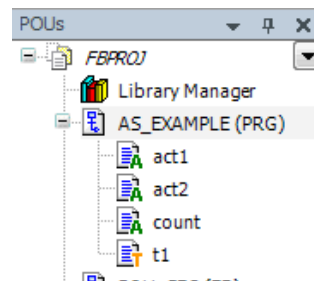


Figura 5-23. Ações na Árvore de POUs

NOTA: Nomes de passo devem ser exclusivos no escopo de uma POU principal. Uma ação não pode ter o mesmo nome do passo ao qual está atribuída.

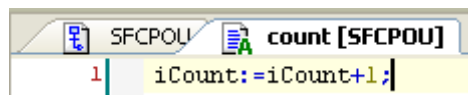


Figura 5-24. Exemplo de uma Ação Escrita em ST

Existem os seguintes tipos de ações:

- Ações IEC
- Ações de passo IEC estendidas

### Ação de Passo em Conformidade com a IEC (Ação IEC)

Esta ação está de acordo com a norma IEC 61131-3 e será processada de acordo com seu qualificador na primeira vez que o passo tornar-se ativo e, uma segunda vez, quando o mesmo for desativado. No caso de haver várias ações atribuídas a um passo (lista de ações) elas serão executadas de cima para baixo.

Diferentes qualificadores podem ser usados para ações de passo IEC (ao contrário de uma ação de passo normal).

Outra diferença para a ação de passo normal é que cada ação de passo IEC possui uma memória de controle que permite garantir que, mesmo que a ação seja chamada por outro passo, ela será executada somente uma vez. Nas ações de passo normais não há garantia em relação a isto.

Uma ação de passo IEC é representada por uma caixa bipartida, conectada à direita de um passo via uma linha de conexão. No lado esquerdo, é exibido o qualificador da ação e, no direito, o nome da ação. Ambos podem ser editados na linha.

Ações de passo IEC são associadas a um passo via comando *Inserir Associação de Ação Após*. Uma ou mais ações podem ser associadas a um passo. A posição da nova ação depende da posição atual do cursor e do comando. As ações devem estar disponíveis no projeto e são inseridas com um nome de ação único (por exemplo, MainPrg.a1).

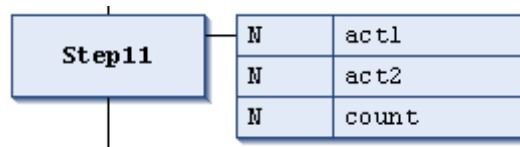


Figura 5-25. Lista de Ações IEC Associadas a um Passo

Cada caixa de ação mostra, na primeira coluna, o qualificador e, na segunda, o nome da ação.

### Ações de Passo IEC Estendidas

Estas ações são uma extensão da norma IEC, anteriormente conhecidas como ações de passo. As ações devem estar disponíveis como objetos no SFC. Os nomes das ações devem ser exclusivos.

### Ação de Entrada de Passo

Este tipo de ação de passo será processado assim que o passo tornar-se ativo e antes da ação de passo ativo.

A ação é associada a um passo via uma entrada no campo *Entrada de Passo* das propriedades do passo. É representada por um “E” no canto inferior esquerdo da caixa de passo.

### Ação de Passo Ativo

Este tipo de ação de passo será processado quando o passo tornar-se ativo e após uma possível ação de entrada de passo ter sido processada. No entanto, ao contrário de uma ação de passo IEC, ela não é executada mais de uma vez quando é desativada e não pode ter qualificadores atribuídos.

A ação é associada a um passo via uma entrada no campo *Passo Ativo* das propriedades do passo. É representada por um pequeno triângulo no canto superior direito da caixa do passo.

### Ação de Saída de Passo

Uma ação de saída será executada uma vez quando o passo for desativado. Observe, entretanto, que esta execução não será feita no mesmo ciclo, mas no começo no ciclo subsequente.

A ação é associada a um passo via uma entrada no campo *Saída de Passo* nas propriedades do passo. É representada por um X no canto inferior direito da caixa de passo.

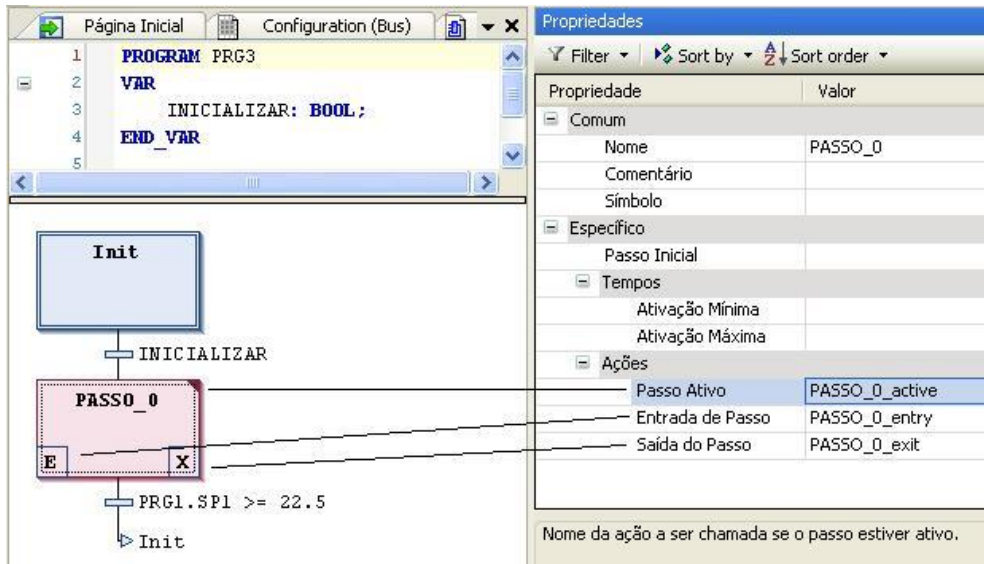


Figura 5-26. Ações de Passo IEC Estendidas

As ações de *Passo Ativo*, *Entrada de Passo* e *Saída de Passo* são definidas nas propriedades do passo.

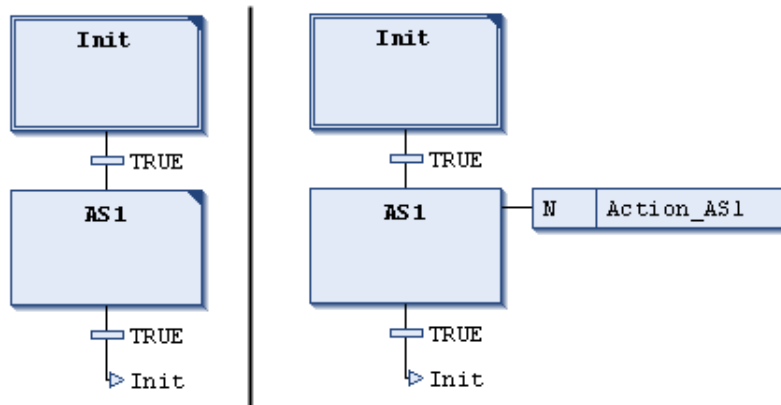


Figura 5-27. Ações de Passo Ativo

### Ramificações

Um sequenciamento gráfico de funções pode divergir, isto é, a linha de processamento pode ser ramificada em duas ou mais linhas (ramificações). Ramificações paralelas serão processadas paralelamente (ambas de uma só vez). As ramificações alternativas dependerão da condição da transição antecessora. Cada ramificação em um gráfico é precedida por uma linha horizontal dupla (paralela) ou simples (alternativa) e também é finalizada por uma linha do mesmo tipo ou por um salto.

### Ramificação Paralela

**Símbolo:**

A ramificação paralela deve iniciar e terminar com um passo. As ramificações paralelas podem conter ramificações alternativas ou outras paralelas.

As linhas horizontais antes e após a área ramificada são duplas.

Processamento no modo online: se a transição precedente (t2 na Figura 5-28) for TRUE, os primeiros passos de todas as ramificações paralelas se tornarão ativos. As ramificações paralelas serão

processadas paralelamente uma à outra antes da transição subsequente (t3 na Figura 5-28) ser considerada.

Uma ramificação paralela é inserida via comando *Inserir Ramificação à Direita* no passo atualmente selecionado.

As ramificações paralelas e alternativas podem ser transformadas através dos comandos *Paralela* e *Alternativa*. Isto pode ser útil durante a programação.

Um rótulo de ramificação é automaticamente adicionado à linha horizontal que precede a ramificação denominada *Ramificação<n>*, onde n é o número de execução, iniciando em zero. Este rótulo pode ser especificado ao definir um destino para o salto.

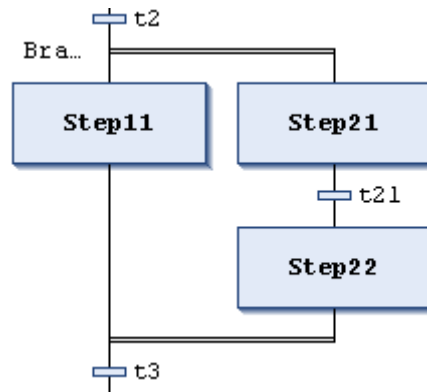



Figura 5-28. Ramificação Paralela

### Ramificação Alternativa

Símbolo: 

Uma ramificação alternativa deve iniciar e terminar com uma transição. As ramificações alternativas podem conter ramificações paralelas ou outras alternativas.

As linhas horizontais antes e após a área ramificada são simples.

Se o passo que precede a linha alternativa de início estiver ativo, então a primeira transição de cada ramificação alternativa será avaliada da esquerda para a direita. A primeira transição a partir da esquerda que apresentar o valor TRUE será aberta, e os passos seguintes serão ativados.

Ramificações alternativas são inseridas via comando *Inserir Ramificação à Direita* quando a transição estiver selecionada.

As linhas horizontais antes e após a área ramificada são simples.

Observe que as ramificações paralelas e alternativas podem ser convertidas através dos comandos *Paralela* e *Alternativa*. Isto pode ser útil durante a programação.



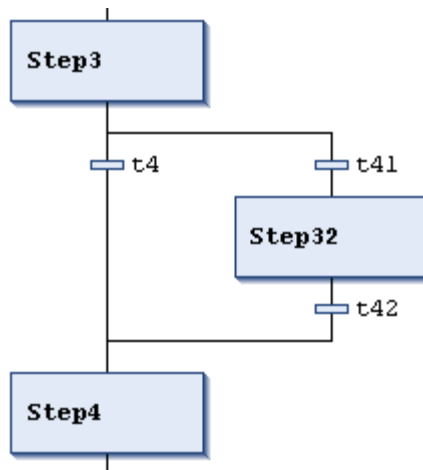


Figura 5-29. Ramificação Alternativa

Salto

Símbolo:

Um salto é representado por uma linha de conexão vertical com uma seta horizontal e pelo nome do destino do salto.

Um salto define o próximo passo a ser processado assim que a transição precedente for TRUE. Saltos podem ser necessários porque as linhas de processamento não devem cruzar ou retornar para cima.

Além do salto padrão no final do gráfico, um salto somente pode ser usado no final de uma ramificação. Ele é inserido via comando *Inserir Salto Após* quando a última transição da ramificação estiver selecionada.

O destino do salto é especificado pela string de texto associada, que pode ser editada na linha. Pode ser o nome de um passo ou o rótulo de uma ramificação paralela.

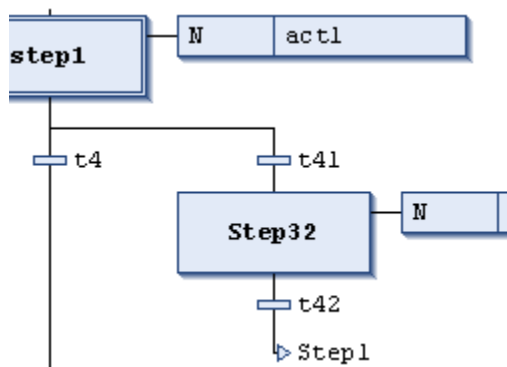


Figura 5-30. Salto

Macro

Símbolo:

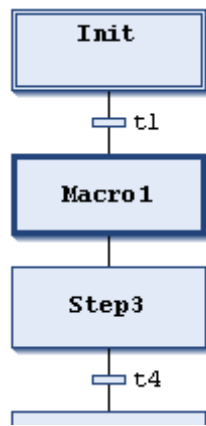


Figura 5-31. Visualização Principal do Editor SFC

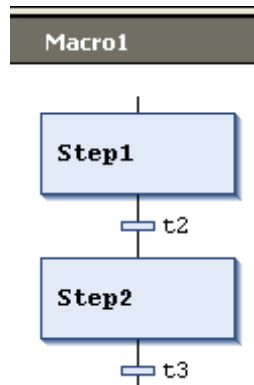


Figura 5-32. Visualização do Editor da Macro para a Macro 1

Uma macro é representada por uma caixa com moldura em negrito contendo o nome da mesma.

Ela inclui uma parte do gráfico SFC que não é visível no editor principal.

O fluxo do processo não é influenciado pelo uso de macros, pois é apenas uma maneira de esconder algumas partes do programa, por exemplo, para simplificar a exibição.

Uma caixa de macro é inserida pelo comando *Inserir Macro* (ou *Inserir Macro Após*). O nome da macro pode ser editado na linha.

Para abrir o editor da macro, execute um clique duplo na sua caixa ou use o comando *Zoom na macro*. Desta forma, é possível editar, assim como na visualização do editor principal, e entrar na seção desejada do gráfico SFC. Para sair, use o comando *Zoom fora da macro*.

A linha de título do editor da macro sempre mostra o caminho da macro no SFC atual.



Figura 5-33. Linha de Título do Editor da Macro

### Qualificador

Para configurar como as ações devem estar associadas aos passos IEC, são inseridos nelas alguns qualificadores no campo correspondente.

Estes qualificadores são tratados pelo bloco funcional SFCActionControl da IecSfc.library, o qual é automaticamente incluído em um projeto através das funcionalidades do SFC.

São eles:

Qualificador	Nome	Descrição
<b>N</b>	Não-armazenado	A ação está ativa enquanto o passo estiver ativo.
<b>R</b>	Reset	A ação é desativada.
<b>S</b>	Set	A ação inicia quando o passo se torna ativo e assim continua até que esta seja desativada.
<b>L</b>	Limitado no tempo	A ação inicia quando o passo se torna ativo e assim continua até que este seja desativado, ou até que um tempo determinado tenha transcorrido.
<b>D</b>	Atrasado no tempo	Um temporizador por atraso inicia quando o passo se torna ativo. Se este se mantiver ativo após o atraso, a ação inicia e continua até que o passo seja desativado.
<b>P</b>	Pulso	A ação inicia quando o passo se torna ativo/inativo e é executada uma vez.
<b>SD</b>	Armazenado e atrasado no tempo	A ação inicia após o tempo de atraso e se mantém até o reset.
<b>DS</b>	Atrasado e armazenado	Se o passo ainda está ativo após o tempo de atraso especificado, a ação inicia e continua até que haja o reset.
<b>SL</b>	Armazenado e limitado no tempo	A ação inicia quando o passo se torna ativo e continua durante o tempo especificado ou até que haja um reset.

**Tabela 5-4. Qualificadores**

Os qualificadores L, D, SD, DS e SL precisam de um valor de tempo no formato da constante TIME.

NOTA: Após a desativação de uma ação IEC, a mesma será executada ainda mais uma vez. Isto significa que cada ação é executada, no mínimo, duas vezes.

### Variáveis Implícitas - Memórias SFC

Cada passo SFC e ação IEC fornece variáveis implicitamente geradas para monitorar os seus status durante a execução. As variáveis também podem ser definidas para monitorar e controlar a execução de um SFC (timeouts, reset, modo auxiliar). Estas variáveis também são geradas implicitamente pelo uso do objeto SFC.

Basicamente, para cada passo e ação IEC, é gerada uma variável implícita. Exemplo: uma instância de estrutura denominada step1 e um passo também denominado step1. Nas propriedades do elemento, é possível definir se uma definição de símbolo deve ser exportada para a configuração de símbolos (para esta memória) e como este símbolo deve ser acessado no CP.

Os tipos de dados para as variáveis implícitas são definidos na biblioteca IecSFC.library. Esta será automaticamente incluída no projeto assim que um objeto SFC for adicionado.

### Status de Passo e Ação e Tempo de Passo

Basicamente, para cada passo e para cada ação IEC, é criada uma variável de estrutura implícita dos tipos SFCStepType e SFCActionType. Os componentes de estrutura (memórias) descrevem o status de um passo e ação ou o tempo de processamento de um passo ativo.

A sintaxe para a declaração da variável implicitamente criada é:

```
<Nome do passo>: SFCStepType;
```

E:

```
_<Nome da ação>: SFCActionType;
```

NOTA: As ações sempre são precedidas por um sublinhado.

Memórias booleanas para estados de passos:

- `<nome do passo>.x`: mostra o status de ativação atual
- `<nome do passo>._x`: mostra o status de ativação para o próximo ciclo

Se `<nome do passo>.x = TRUE`, o passo será executado no ciclo atual.

Se `<nome do passo>._x = TRUE` e `<nome do passo>.x = FALSE`, o passo será executado no ciclo seguinte, ou seja, `<nome do passo>._x` será copiado para `<nome do passo>.x` no início de um ciclo.

Memórias booleanas para estados de ações:

- `_<nome da ação>.x` é TRUE, se a ação é executada
- `_<nome da ação>._x` é TRUE, se a ação está ativa

### Geração de Símbolo

Nas propriedades do elemento de um passo ou ação, é possível definir se uma definição de símbolo deve ser adicionada a uma aplicação de símbolo possivelmente criada e enviada (em relação à memória do nome do passo ou da ação). Para tal, selecione o item para o direito de acesso desejado na coluna *Símbolo* da visualização das propriedades do elemento.

NOTA: As memórias descritas acima podem ser usadas para forçar um determinado status de valor para um passo, ou seja, defini-lo como ativo. No entanto, esteja ciente de que isto pode causar descontrole no SFC.

### Tempo Via Variáveis TIME

A memória “t” determina o tempo atual transcorrido desde que o passo se tornou ativo. Isto ocorre apenas para passos, independentemente do tempo configurado nos atributos de passo (veja abaixo as considerações sobre o SFCError).

Para passos:

`<nome do passo>.t` (`<nome do passo>._t` não pode ser usado para fins externos)

Para ações: as variáveis de tempo implícitas não são usadas.

### Controle de Execução SFC (Timeouts, Reset, Modo Auxiliar)

Algumas variáveis implicitamente disponíveis, também denominadas memórias SFC podem ser usadas para controlar a operação de um SFC, por exemplo, para indicar overflows de tempo ou habilitar o modo auxiliar para trocar transições.

Para acessar estas memórias e colocá-las em funcionamento, elas devem ser declaradas e ativadas. Isto deve ser feito no diálogo de *Configurações SFC* (subdiálogo do diálogo *Propriedades* do objeto). A declaração manual só é necessária para habilitar o acesso de escrita a partir de outra POU (veja abaixo, **Acessando Memórias**). Entretanto, neste caso, considere: ao declarar a memória globalmente, a opção Declarar, no diálogo das *Configurações SFC*, deve estar desativada para não obter uma memória local implicitamente declarada (que então seria usada, em vez da global). As configurações SFC para uma POU SFC, inicialmente, são determinadas pelas definições atualmente configuradas no diálogo *Opções SFC*.

Observe que a declaração de uma variável de memória feita somente através do diálogo *Configurações SFC* somente será visível na visualização online da POU SFC.

Exemplo do uso do SFCError: há uma POU SFC nomeada `sfc1` contendo um passo `s2`, com limites de tempo definidos nos atributos de passo. Veja os atributos exibidos na Figura 5-34.

Se, por alguma razão, o passo s2 permanece ativo por um tempo maior que o permitido por suas propriedades (overflow de tempo), uma memória SFC, a qual pode ser lida pela aplicação, pode ser definida.

Para permitir este acesso, declare a memória no diálogo *Configurações SFC*. Para tal, selecione sfc1 na janela *Dispositivos e POU* e escolha o comando *Propriedades* a partir do menu de contexto. Abra o subdiálogo *Configurações SFC* e, na guia *Memórias*, marque as colunas *Declarar* e *Usar* para a memória *SFCError*. Escolher somente *Declarar* deixaria a variável visível na visualização online da parte de declaração de sfc1, porém isto não teria nenhuma função.

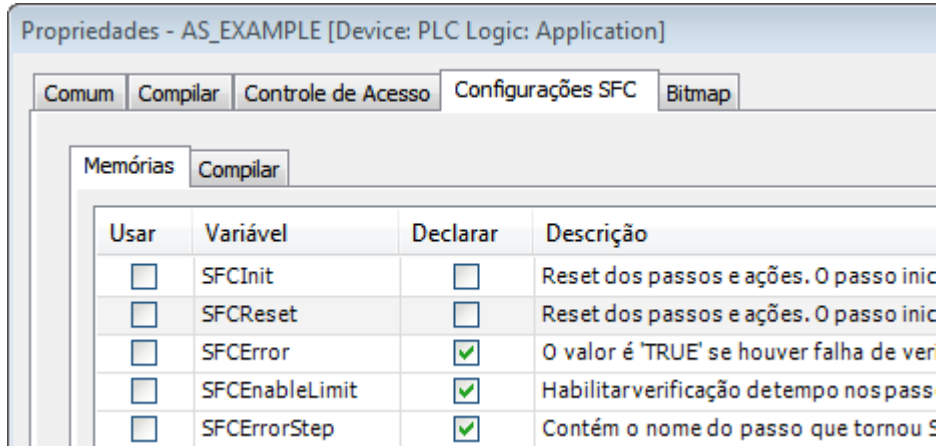


Figura 5-34. Configurações SFC

Leia o SFCError no SFC, por exemplo, em uma ação via SFCError, ou a partir de outra POU via sfc1.SFCError.

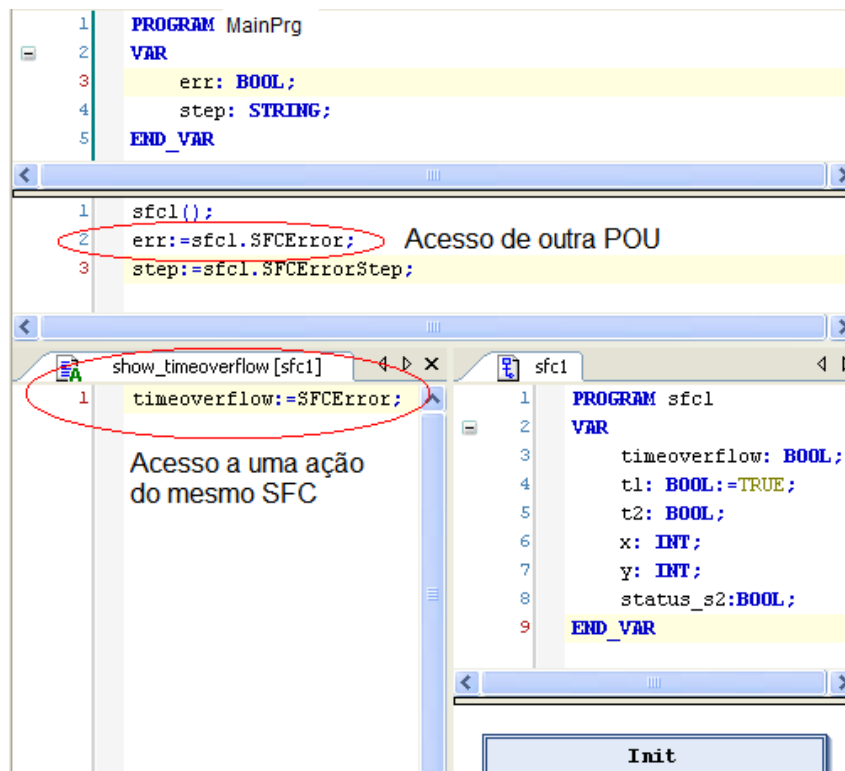


Figura 5-35. Acessando SFCError

SFCError se tornará TRUE assim que ocorrer um timeout em sfc2.

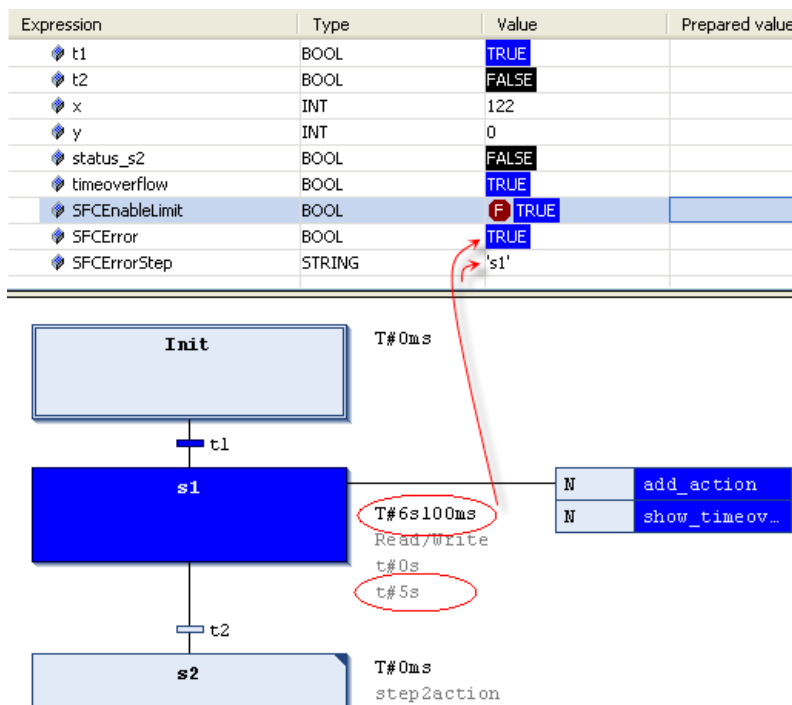


Figura 5-36. Visualização Online do SFC sfc1

As seguintes variáveis implícitas (memórias) podem ser usadas. Para tanto, elas devem estar declaradas e ativadas nas *Configurações SFC*:

Variável	Descrição
<b>SFCInit: BOOL;</b>	Se a variável for TRUE, o sequenciamento gráfico de função será configurado para o passo inicial. Todos os passos, ações e outras memórias SFC serão inicializados. O passo inicial permanecerá ativo, mas não será executado enquanto a variável for TRUE. SFCInit deve ser redefinida para FALSE para retomar o processamento normal.
<b>SFCReset: BOOL;</b>	Esta variável comporta-se de maneira similar a SFCInit. Diferentemente desta, no entanto, outros processamentos ocorrem após a inicialização do passo inicial. Assim, neste caso por exemplo, poderia ser feita uma atribuição FALSE na memória SFCReset no passo inicial.
<b>SFCError: BOOL;</b>	Assim que ocorrer um timeout em um dos passos no SFC, a variável será TRUE. Pré-requisito: SFCEnableLimit deve ser TRUE. Observe que qualquer outro timeout não poderá ser registrado até que SFCError seja FALSE. SFCError deve ser definida para usar a outra memória de controle de tempo (SFCErrorStep, SFCErrorPOU e SFCQuitError).
<b>SFCEnableLimit: BOOL;</b>	Esta variável pode ser usada para a ativação explícita (TRUE) e desativação (FALSE) do controle de tempo nos passos via SFCError. Isto significa que, se esta variável for declarada e ativada (Configurações SFC), então ela deve ser configurada para TRUE para o funcionamento de SFCError. Caso contrário, quaisquer timeouts dos passos não serão registrados. Isto pode ser interessante durante as inicializações ou na operação manual. Se a variável não está definida, SFCError funcionará automaticamente. Naturalmente, como pré-requisito, SFCError deve estar definida.
<b>SFCErrorStep: STRING;</b>	Esta variável armazena o nome de um passo no qual um timeout foi registrado pelo SFCError.timeout. Pré-requisito: SFCError deve estar definida.
<b>SFCErrorPOU: STRING;</b>	Esta variável armazena o nome da POU SFC na qual ocorreu um timeout. Pré-requisito: SFCError deve estar definida.
<b>SFCQuitError: BOOL;</b>	Enquanto esta variável for TRUE, a execução do diagrama SFC é interrompida e a variável SFCError será FALSE. Assim que esta se tornar FALSE, todos os estados de tempo atuais nos passos ativos serão FALSE. Pré-requisito: SFCError deve estar definida.
<b>SFCPause: BOOL;</b>	Enquanto esta variável for TRUE, a execução do diagrama SFC é interrompida.

<b>SFCTrans: BOOL;</b>	Esta variável torna-se TRUE assim que houver uma transição.
<b>SFCCurrentStep: STRING;</b>	Esta variável armazena o nome do passo atualmente ativo, independentemente da monitoração do tempo. Em caso de sequências simultâneas, o nome do passo à direita será registrado.
<b>SFCTip, SFCTipMode: BOOL;</b>	Estas variáveis permitem o modo "inching" no gráfico atual. Quando este modo foi ativado pelo SFCTipMode=TRUE, somente será possível pular para o próximo passo configurando SFCTip=TRUE (borda de subida). Enquanto SFCTipMode for configurado para FALSE, será possível pular pelas transições.

Tabela 5-5. Variáveis Implícitas

Um timeout foi detectado no passo s1 no objeto POU do SFC pela memória SFCErrror.

The screenshot displays the 'Device.Application.POU' memory table with the following data:

Expressão	Tipo	Valor	Valor Preparado
t2111	BOOL	FALSE	
t222	BOOL	FALSE	
SFCErrror	BOOL	TRUE	
SFCErrrorPOU	STRING	'POU'	
SFCErrrorStep	STRING	's1'	
SFCErrrorQuitError	BOOL	FALSE	

Below the table, a step diagram shows step 's1' (blue box) with a transition 'N act1' (blue box). The step 's1' is active, indicated by a 'TRUE' label above it. The step details are:

```

T#2h57m54s995ms
This is Step s1.
Minimal active: t#2s
Maximal active: t#4s
Step active: act1

```

Figura 5-37. Exemplo de Algumas Memórias SFCErrror no Modo Online do Editor

### Acessando Memórias

Para habilitar o acesso às memórias para o controle da execução SFC (timeouts, reset, modo auxiliar), as variáveis de memória devem ser declaradas e ativadas conforme descrito acima (Controle de execução SFC).

Sintaxe para acesso:

- A partir de uma ação ou transição na POU SFC: <nome do passo>.<flag> e <nome da ação>.<flag>. Exemplos: status:=step1.\_x; checkerror:=SFCErrror
- A partir de outra POU : <SFC POU>.<nome do passo>.<flag> e <SFC POU>.<nome da ação>.<flag>. Exemplos: status:=SFC\_prog.step1.\_x; checkerror:=SFC\_prog.SFCErrror;

No caso de acesso de escrita a partir de outra POU, a variável implícita também deve estar declarada explicitamente como uma variável VAR\_INPUT da POU SFC, ou globalmente, por exemplo, em uma GVL.

Exemplo:

Declaração local:

```
PROGRAM SFC_prog
VAR_INPUT
SFCinit:BOOL;
END_VAR
```

Ou declaração global em uma GVL:

```
VAR_GLOBAL
SFCinit:BOOL;
END_VAR
```

Acessando a memória no MainPrg:

```
PROGRAM MainPrg
VAR
setinit: BOOL;
END_VAR
SFC_prog.SFCinit:=setinit; //acesso de escrita em SFCinit no SFC_prog.
```

### Sequência de Processamento no SFC

No modo online, os tipos de ação específicos serão processados de acordo com uma sequência definida (veja na Tabela 5-6).

Primeiramente, observe a terminologia:

- Passo ativo: é o passo no qual a ação está sendo executada. No modo online, os passos ativos são exibidos na cor azul.
- Passo inicial: no primeiro ciclo após a POU SFC ter sido chamada, o passo inicial automaticamente torna-se ativo e a ação do passo associada é executada.
- Ações IEC: são executadas no mínimo duas vezes (a primeira vez quando são ativadas e a segunda, no ciclo seguinte, quando são desativadas).
- Ramificações alternativas: se o passo anterior à linha horizontal das ramificações estiver ativo, então a primeira transição de cada ramificação específica será avaliada, da esquerda para a direita. A primeira transição à esquerda que apresentar o valor TRUE provocará a execução da ramificação respectiva, ou seja, o passo subsequente na ramificação se tornará ativo.
- Ramificações paralelas: se a linha dupla no início de uma ramificação paralela estiver ativa e a condição de transição precedente apresentar o valor TRUE então, em todas as ramificações paralelas, os passos subsequentes serão ativados. As ramificações serão processadas paralelamente. O passo seguinte à linha dupla, ao final da ramificação, se tornará ativo quando todos os passos anteriores estiverem ativos e a condição de transição após a linha dupla apresentar o valor TRUE.

Ordem de processamento de elementos em uma sequência:

Item	Descrição
<b>Reset</b>	Todas as memórias de controle das ações IEC são resetadas (exceto as memórias das ações IEC chamadas nas próprias ações).
<b>Ações de saída de passo</b>	Todos os passos são verificados na ordem do sequenciamento gráfico (de cima para baixo e da esquerda para a direita) para determinar se o requisito para a execução da ação de saída de passo foi fornecido e se será executada. Uma ação de saída será executada se o passo for desativado, ou seja, sua entrada e suas ações de passo foram executados durante o último ciclo, e se a transição para o passo seguinte for TRUE.
<b>Ações de entrada de passo</b>	Todos os passos são testados na ordem do sequenciamento para determinar se o requisito para a execução da ação de entrada de passo foi fornecido e, se este for o caso, se esta será executada. Uma ação de entrada será executada se o passo precedente à transição for TRUE e, assim, o passo tiver sido ativado.



<b>Verificação de timeout, Ações ativas de passo</b>	Para todos os passos, na ordem do sequenciamento.
<b>Ações IEC</b>	As Ações IEC usadas no sequenciamento são executadas em ordem alfabética. Isto é feito em duas etapas na lista de ações. Primeiramente, são executadas todas as ações IEC desativadas no ciclo atual e, em seguida, todas as ações IEC ativas.
<b>Verificação de transição, ativando os próximos passos</b>	Transições são avaliadas: se o passo no ciclo atual estava ativo e a transição seguinte retornar TRUE (e, se aplicável, o tempo ativo mínimo já tiver transcorrido), então o passo seguinte será ativado.

Tabela 5-6. Ordem de Processamento de Elementos em uma Sequência

## NOTAS:

- Em relação à implementação das ações, considere que uma ação pode ser transmitida várias vezes em um ciclo por estar associada a múltiplas sequências. Exemplo: um SFC pode ter duas ações IEC (A e B), sendo ambas implementadas no SFC e ambas chamando a ação IEC C. Deste modo, as ações A e B podem estar ativas no mesmo ciclo e, posteriormente, em ambas as ações, a ação C pode estar ativa. Neste caso, C teria sido chamada duas vezes. Se a mesma ação IEC for usada simultaneamente em diferentes níveis de um SFC, isto poderia causar efeitos indesejados devido ao sequenciamento do processamento descrito acima. Por isto, uma mensagem de erro é apresentada.

- Considere a possibilidade de usar variáveis implícitas para controlar o status dos passos, das ações e a execução do gráfico.

**Editor SFC no Modo Online**

No modo online, o editor SFC fornece visualizações para monitoração, escrita e forçamento de variáveis e expressões no controlador. Veja:

- Para obter informações sobre como abrir objetos no modo online, **consulte Interface do Usuário no Modo Online** no Manual de Utilização MasterTool IEC XE – MU299048.
- A janela do editor de um objeto SFC também inclui o *Editor de Declaração* na parte superior. Para obter informações gerais sobre o editor de declaração no modo online, consulte **Editor de Declaração no Modo Online** no Manual de Utilização MasterTool IEC XE – MU299048. No caso de haver variáveis implícitas declaradas (memórias SFC) através do diálogo de *Configurações do SFC*, estas serão ali adicionadas, porém não serão visualizadas no modo offline do *Editor de Declaração*.
- Considere também a sequência de processamento dos elementos de um Sequenciamento Gráfico de Funções.
- Consulte as informações sobre as propriedades do objeto, as opções do editor SFC e os padrões SFC para obter informações sobre as configurações referentes à compilação e exibição online de elementos SFC e seus atributos.
- Considere a possibilidade de usar memórias para monitorar e controlar o processamento de um SFC.

*Monitoração*

Passos ativos são exibidos com a cor azul. A exibição dos atributos de passo depende das opções do editor SFC atualmente configuradas.

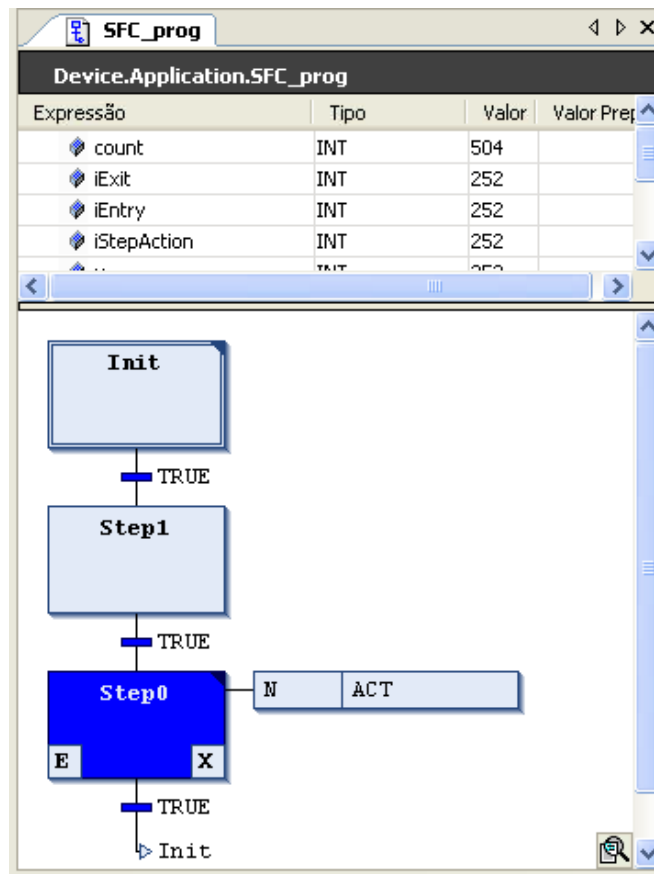


Figura 5-38. Visualização Online de um Objeto do Tipo Programa SFC\_prog

## Texto Estruturado (ST) / Texto Estruturado Estendido (ExST)

Texto estruturado é uma linguagem de programação textual de alto nível, similar a PASCAL ou C. O código de programa é composto de expressões e instruções. Ao contrário da IL (Lista de Instruções), inúmeras construções podem ser usadas para laços de programação, permitindo, assim, o desenvolvimento de complexos algoritmos.

Exemplo:

```
IF value < 7 THEN
WHILE value < 8 DO
value:=value+1;
END_WHILE;
END_IF;
```

Texto Estruturado Estendido (ExST) é uma extensão da norma IEC 61131-3 específica do MasterTool IEC XE para Texto Estruturado (ST). Exemplos: Atribuição como expressão e operadores Set/Reset.

## Expressões

Uma expressão é uma construção que retorna um valor após sua avaliação. Este valor é usado nas instruções.

Expressões são compostas de operadores, operandos e/ou atribuições. Um operando pode ser uma constante, uma variável, uma chamada de função ou outra expressão.

Exemplos:

Expressão	Definição
33	Constante
ivar	Variável
fct(a,b,c)	Chamada de função
a AND b	Expressão
(x*y) / z	Expressão
real_var2 := int_var;	Atribuição

**Tabela 5-7. Expressões**

### Avaliação de Expressões

A avaliação da expressão ocorre através do processamento dos operadores, de acordo com certas regras de ligação. O operador com a ligação mais forte é processado primeiro, seguido pelo segundo mais forte e assim por diante, até que todos os operadores tenham sido processados.

Operadores com a mesma força de ligação são processados da esquerda para a direita.

Abaixo é apresentada uma tabela de operadores ST na ordem de sua força de ligação, da mais forte para a mais fraca.

Operação	Símbolo
Colocar entre parênteses	(expressão)
Chamada de função	Nome da função (lista de parâmetros)
Exponenciação	EXPT
Negar	NOT
Complementos	
Multiplicar	*
Dividir	/
Módulo	MOD
Somar	+
Subtrair	-
Comparar	<, >, <=, >=
Igual a	=
Diferente de	<>
Booleano AND	AND
Booleano XOR	XOR
Booleano OR	OR

**Tabela 5-8. Operadores ST**

### Atribuição como Expressão

Como extensão da norma IEC 61131-3 (ExST), o MasterTool IEC XE permite que atribuições sejam usadas como expressões.

Exemplos:

Expressões	Comentário
int_var1 := int_var2 := int_var3 + 9;	Atribuição do resultado de uma expressão para int_var2 e int_var1
real_var1 := real_var2 := int_var;	Atribuições corretas, real_var1 e real_var2 assumirão o valor de int_var
int_var := real_var1 := int_var;	Isto levará a uma mensagem de erro devido à não correspondência dos tipos real e int
IF b := (i = 1) THEN	Atribuição errada, pois está sendo usado o

<code>i := i + 1; END_IF</code>	operador de atribuição para comparar duas expressões.
-------------------------------------	---

Tabela 5-9. Atribuições Usadas como Expressões

## Instruções

Instruções definem as ações a serem realizadas nas expressões. As seguintes instruções podem ser usadas em ST.

Tipo de instrução	Exemplo
Atribuição	<code>A:=B; CV := CV + 1; C:=SIN(X);</code>
Chamando um bloco funcional e uso da saída FB	<code>CMD_TMR(IN := %IX5, PT := 300); A:=CMD_TMR.Q</code>
RETURN	<code>RETURN;</code>
IF	<code>D:=B*B; IF D&lt;0.0 THEN C:=A; ELSIF D=0.0 THEN C:=B; ELSE C:=D; END_IF;</code>
CASE	<code>CASE INT1 OF 1: BOOL1 := TRUE; 2: BOOL2 := TRUE; ELSE   BOOL1 := FALSE;   BOOL2 := FALSE; END_CASE;</code>
FOR	<code>J:=101; FOR I:=1 TO 100 BY 2 DO   IF ARR[I] = 70 THEN     J:=I;     EXIT;   END_IF; END_FOR;</code>
WHILE	<code>J:=1; WHILE J&lt;= 100 AND ARR[J] &lt;&gt; 70 DO   J:=J+2; END_WHILE;</code>
REPEAT	<code>J:=-1; REPEAT   J:=J+2; UNTIL J= 101 OR ARR[J] = 70 END_REPEAT;</code>
EXIT	<code>EXIT;</code>
CONTINUE	<code>CONTINUE;</code>
JMP	<code>label: i:=i+1; JMP label;</code>
Instrução vazia	<code>;</code>

Tabela 5-10. Instruções

### Operador de Atribuição

À esquerda de uma atribuição, deve haver um operando (variável, endereço) ao qual o valor da expressão do lado direito é atribuído usando o operador de atribuição “:=”.

Veja também o operador **MOVE**, que apresenta a mesma função.

Exemplo:

```
Var1 := Var2 * 10;
```

Após a execução desta linha, Var1 representará dez vezes o valor de Var2.

### Características Estendidas

Operadores de atribuição que não fazem parte da norma 61131-3 (ExST)

Operador Set: o valor uma vez definido como TRUE, assim permanecerá.

Exemplo:

```
a S= b;
```

O operando “a” assume o valor de “b” uma vez em TRUE, ele assim permanecerá, mesmo se “b” tornar-se FALSE novamente.

Operador Reset: uma vez definido como FALSE, o operando assim permanecerá.

Exemplo:

```
a R= b;
```

O operando “a” é configurado para FALSE quando b= TRUE.

#### NOTA:

No caso de uma atribuição múltipla, todas as atribuições Set e Reset referem-se ao último membro da atribuição. Exemplo: a S= b R= fun1(par1,par2);

Neste caso, “b” assume o valor da saída de fun1, Mas “a” não assume o valor de “b”, e sim o valor configurado da saída fun1.

Uma atribuição pode ser usada como uma expressão.

### Chamando Blocos Funcionais em ST

Um bloco funcional (FB) é chamado, na linguagem Texto Estruturado, de acordo com a sintaxe abaixo:

```
<nome da instância do FB>(variável de entrada FB:=<valor ou endereço>|,  
<outra variável de entrada FB:=<valor ou endereço>|...outras variáveis de  
entrada FB);
```

Exemplo:

Suponha que um bloco funcional de temporização (TON) seja chamado com atribuições para os parâmetros IN e PT.

A variável resultante Q é atribuída à variável A. O FB é instanciado por “TMR:TON;”.

A variável resultante, como em IL, é endereçada em conformidade com a sintaxe <nome da instância FB>.<variável FB>:

```
TMR(IN := %IX5, PT := 300);  
A:=TMR.Q
```

### Instrução RETURN

A instrução RETURN pode ser usada para sair de uma POU, por exemplo, dependendo de uma condição.

Sintaxe:

```
RETURN;
```

Exemplo:

```
IF b=TRUE THEN
```

```
RETURN;  
END_IF;  
a:=a+1;
```

Se “b” for TRUE, a instrução a:=a+1 não será executada e a POU será deixada imediatamente.

### *Instrução IF*

A instrução IF permite verificar uma condição e, dependendo desta, executar instruções.

**Sintaxe:**

```
IF <Expressão booleana 1> THEN  
<Instruções IF>  
{ELSIF <Expressão booleana 2> THEN  
<Instruções ELSIF 1>  
...  
ELSIF <Expressão booleana 2> THEN  
<Instruções ELSIF n-1>  
ELSE  
<Instruções ELSE>}  
END_IF;
```

A parte entre chaves é opcional.

Se a <Expressão booleana 1> retornar TRUE, então somente as <Instruções IF> serão executadas (as outras instruções não).

Caso contrário, as expressões booleanas, começando com a <Expressão booleana 2>, são avaliadas uma após a outra até que uma das expressões retorne TRUE. Assim, somente aquelas instruções após esta expressão booleana e antes de ELSE ou ELSIF serão avaliadas.

Se nenhuma das expressões booleanas retornar TRUE, então somente as <Instruções ELSE> serão avaliadas.

**Exemplo:**

```
IF temp<17 THEN  
heating_on := TRUE;  
ELSE  
heating_on := FALSE;  
END_IF;
```

Neste caso, a variável HEATING\_ON é acionada quando a TEMP for menor 17. Caso contrário, HEATING\_ON será FALSE.

### *Instrução CASE*

Com as instruções CASE, uma instrução pode combinar várias instruções condicionadas pela a mesma variável de condição, em uma construção específica.

**Sintaxe:**

```
CASE <Var1> OF  
<Valor 1>: <Instrução 1>  
<Valor 2>: <Instrução 2>  
<Valor 3, Valor 4, Valor 5>: <Instrução 3>  
<Valor 6 .. Valor 10>: <Instrução 4>  
...  
<Valor n>: <Instrução n>  
ELSE  
< instrução ELSE >  
END_CASE;
```

Uma instrução CASE é processada de acordo com o seguinte modelo:

- Se a variável em <Var1> apresentar o valor <Valor 1>, então a instrução <Instrução 1> será executada.
- Se <Var 1> não apresentar nenhum dos valores indicados, então a <instrução ELSE> será executada.
- Se a mesma instrução deve ser executada para muitos valores de variáveis, estas escrevem os valores um após o outro (separados por vírgulas), condicionando, assim, uma execução em comum.
- Se uma instrução deve ser executada em um intervalo de valores das variáveis, estas escrevem o valor inicial e o final separados por dois pontos. Assim, determina-se uma condição em comum.

Exemplo:

```
CASE INT1 OF
1, 5: BOOL1 := TRUE;
      BOOL3 := FALSE;
2:    BOOL2 := FALSE;
      BOOL3 := TRUE;
10..20: BOOL1 := TRUE;
        BOOL3:= TRUE;
ELSE
      BOOL1 := NOT BOOL1;
      BOOL2 := BOOL1 OR BOOL2;
END_CASE;
```

### Laço FOR

O laço FOR permite a repetição de processos.

Sintaxe:

```
FOR <INT_VAR>:=<VALOR_INICIAL> TO <VALOR_FINAL> {BY <TAMANHO DO PASSO>} DO
<INSTRUÇÕES>
END_FOR;
```

<INSTRUÇÕES> será executado enquanto o contador <INT\_VAR> não for maior que <VALOR\_FINAL>. Isto será verificado antes que <INSTRUÇÕES> seja executado, de forma que nunca haverá uma execução se esta condição não for atendida.

Quando <INSTRUÇÕES> é executado, <INT\_VAR> é incrementado pelo <TAMANHO DO PASSO> (o qual pode ter qualquer valor inteiro). Se este não existir, o valor será configurado para 1. O laço também será finalizado quando <INT\_VAR> se tornar maior do que <VALOR\_FINAL>.

Exemplo:

```
FOR Counter:=1 TO 5 BY 1 DO
Var1:=Var1*2;
END_FOR;
Erg:=Var1;
```

Suponha que valor inicial de Var1 seja “1”. Então, ela terá o valor “32” após o laço FOR.

NOTA: Se <VALOR\_FINAL> for igual ao valor limite do contador <INT\_VAR> (por exemplo Counter - usado no exemplo mostrado acima declarado como tipo SINT) e se <VALOR\_FINAL> for 127, haverá um laço sem fim. Neste caso, <VALOR\_FINAL> não deve ser igual ao valor limite do contador.

A instrução CONTINUE pode ser usada em um laço FOR.

### Laço WHILE

O laço WHILE pode ser usado da mesma forma que o FOR, apenas com a diferença de que a condição de ruptura pode ser qualquer expressão booleana. Em outras palavras: indica-se uma condição e, quando esta é atendida, o laço é executado.

Sintaxe:

```
WHILE <Expressão booleana> DO
<Instruções>
END_WHILE;
```

<INSTRUÇÕES> será executado repetidamente enquanto <EXPRESSÃO BOLEANNA> retornar TRUE. Se <EXPRESSÃO BOLEANNA> for FALSE na primeira avaliação, então <INSTRUÇÕES> não será executado. Se <EXPRESSÃO BOLEANNA> nunca assumir o valor FALSE, então <INSTRUÇÕES> será repetido indefinidamente, fazendo a aplicação travar, e por consequência, causar a entrada em cão-de-guarda em função do estouro deste parâmetro configurado na tarefa correspondente.

NOTA: O programador deve certificar-se de que não foi gerado um laço infinito. Esta verificação é feita alterando-se a condição na parte de instruções do laço, por exemplo, incrementando um contador para cima ou para baixo e usando-o como condição.

Exemplo:

```
WHILE Counter<>0 DO
    Var1 := Var1*2;
    Counter := Counter-1;
END_WHILE
```

Os laços WHILE e REPEAT são, de certa forma, mais poderosos que o laço FOR, uma vez que não precisam saber o número de ciclos antes da execução do laço. Em alguns casos, portanto, somente é possível trabalhar com estes dois tipos de laços. Se, entretanto, o número de ciclos de laço estiver definido, um laço FOR é preferível, já que este não permite laços infinitos.

A instrução CONTINUE pode ser usada em um laço WHILE.

### Laço REPEAT

O laço REPEAT é diferente do WHILE, pois a condição de quebra é verificada somente após o laço ter sido executado. Isto significa que a execução ocorrerá pelo menos uma vez, independentemente desta condição.

Sintaxe:

```
REPEAT
<Instruções>
UNTIL <Expressão booleana>
END_REPEAT;
```

<INSTRUÇÕES> é executado até que <EXPRESSÃO BOLEANNA> retorne TRUE.

Se a <EXPRESSÃO BOLEANNA> retornar TRUE na primeira execução, então <INSTRUÇÕES> será executado apenas uma vez. Se a <EXPRESSÃO BOLEANNA> não assumir nunca o valor TRUE, então <INSTRUÇÕES> será repetido indefinidamente, fazendo a aplicação travar e por consequência causando a entrada em cão-de-guarda em função do estouro deste parâmetro configurado na tarefa correspondente.

NOTA: O programador deve certificar-se de que não foi gerado nenhum laço infinito. Esta verificação é feita alterando-se a condição na parte de instruções do laço, por exemplo, incrementando um contador para cima ou para baixo e usando-o como condição.

Exemplo:

```
REPEAT
Var1 := Var1*2;
    Counter := Counter-1;
UNTIL
    Counter=0
```



END\_REPEAT:

A instrução CONTINUE pode ser usada em um laço REPEAT.

### *Instrução CONTINUE*

Como uma extensão da norma IEC 61131-3 (ExST), a instrução CONTINUE é suportada em FOR, WHILE e laços REPEAT.

CONTINUE faz com que a execução continue no próximo ciclo de laço.

Exemplo:

```
FOR Counter:=1 TO 5 BY 1 DO
INT1:= INT1/2;
IF INT1=0 THEN
CONTINUE; (* Para evitar a divisão por zero *)
END_IF
Var1:=Var1/INT1; (* Somente será executada, se INT1 não for "0" *)
END_FOR;
Erg:=Var1;
```

### *Instrução EXIT*

Se a instrução EXIT for utilizada em FOR, WHILE ou laço REPEAT, o laço mais interno será finalizado, independentemente da condição definida.

### *Instrução JMP*

A instrução JMP pode ser usada como um salto incondicional para uma linha de código marcada por um rótulo de salto.

Sintaxe:

```
<rótulo>:
JMP <rótulo>;
```

<RÓTULO> é um identificador qualquer, porém único, localizado no início de uma linha do programa. A instrução JMP deve ser seguida pela indicação do destino do salto (deve ser igual ao rótulo pré-definido). Quando a instrução JMP é avaliada, ocorre um salto no processamento para a linha rotulada no programa.

NOTA: O programador deve evitar a criação de laços infinitos, por exemplo, submetendo o salto a uma condição IF.
--

Exemplo:

```
i:=0;
label1: i:=i+1;
(*Instruções*)
IF (i<10) THEN
JMP label1;
END_IF
```

Enquanto a variável “i”, inicializada com zero, for menor que 10, a instrução de salto condicional do exemplo acima provocará repetidos saltos para a linha do programa identificada com LABEL1 e, portanto, causará um processamento repetido das instruções compreendidas entre o rótulo e a instrução JMP. Uma vez que estas instruções incluem também o incremento da variável “i”, pode-se assegurar que a condição de salto resultará FALSE na nona verificação e que, na sequência, o fluxo do programa continuará normalmente.

Esta funcionalidade pode ser alcançada usando laços WHILE ou REPEAT no exemplo. Geralmente, as instruções de salto podem e devem ser evitadas, pois reduzem a legibilidade do código.

### Comentários em ST

Há duas formas de se escrever comentários em um objeto de texto estruturado.

- Iniciar o comentário com “(\*)” e fechá-lo com “\*)”. Isto permite comentários que abrangem uma ou mais linhas. Exemplo:

```
(* Isto é um comentário*)
```

- Comentários de linha única são uma extensão da norma IEC 61131-3: “//” denota o início de um comentário que termina no final da linha. Exemplo:

```
// Este é um comentário.
```

Os comentários podem estar localizados em qualquer lugar, nas partes de declaração ou implementação do editor ST.

Comentários aninhados: comentários podem estar localizados em outros comentários.

Exemplo:

```
(*  
a:=inst.out; (* verificação *)  
b:=b+1;  
*)
```

Neste exemplo, o comentário que inicia no primeiro parênteses não é fechado pelo parênteses seguinte, apenas pelo último.

### Editor ST

O editor ST é usado para criar objetos de programação na linguagem IEC Texto Estruturado (ST) e Texto Estruturado Estendido. Esta última inclui algumas extensões à norma IEC 61131-3.

O editor ST é um editor textual e a sua configuração no diálogo *Opções* pode ser usada para configurar o comportamento e aparência. Ali, se definem as configurações personalizadas para realce de cor, números de linha, guias, recuos e outras.

É possível realizar uma seleção de bloco pressionando-se <ALT> ao selecionar a área de texto desejada (via mouse).

O editor estará disponível na parte inferior da janela, que também inclui o *Editor de Declaração* na parte superior.

Nos casos de erros sintáticos durante a edição, mensagens correspondentes serão exibidas na janela de *Mensagem(s) de Pré-Compilação*. Uma atualização desta janela ocorrerá a cada vez que o foco da janela do editor for reajustado (por exemplo posicionando o cursor em outra janela e, em seguida, retornando à janela do editor).

### Editor ST no Modo Online

No modo online, o Editor de Texto Estruturado (editor ST) fornece visualizações para monitoração e para escrita/forçamento das variáveis e expressões no controlador. A funcionalidade de depuração (breakpoints, passo a passo, etc.) está disponível.

Para informações sobre como abrir objetos no modo online, consulte o item **Interface do Usuário no Modo Online** no Manual de Utilização MasterTool IEC XE – MU299048.

Para informações sobre como inserir valores preparados para variáveis no modo online, consulte o item **Forçamento de Variáveis**.

Observe que a janela do editor de um objeto ST também inclui o Editor de Declaração na parte superior. Para informações sobre o editor de declaração no modo online, consulte o **item Editor de Declaração no Modo Online** no Manual de Utilização MasterTool IEC XE – MU299048.

### Monitoração

Se a monitoração em linha não for explicitamente desativada no diálogo *Opções*, pequenas janelas de monitoração serão exibidas em cada variável, mostrando o valor atual destas (monitoração em linha).

Expressão	Tipo	Valor	Valor Preparado
ivar	INT	27028	
fbinst	FB1		
in	INT	11	
out	INT	11	
fbvar	INT	0	
fbinst2	FB1		
erg	INT	22	

```

1  ivar[27028] := ivar[27028]+1; (* counter *)
2  fbinst(in[11] :=11); (* call function block FB1, input parameter
3  erg[22] :=fbinst.out[11] ; (* read result from FB1 output "out"
4  fbinst2(in[22] :=22); (* call function block FB1, input paramete.
5  erg[22] :=fbinst2.out[22] ; (* read result from FB1 output "out"

```

**Figura 5-39. Visualização Online de um Objeto do Tipo Programa (MainPrg)**

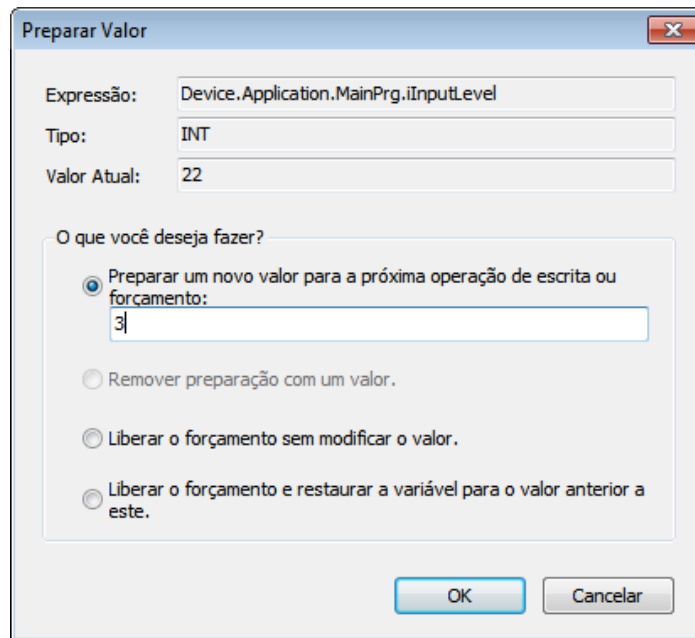
Em uma visualização online de uma POU do tipo bloco funcional, os valores não serão visualizados e, em vez deles, o termo <Valor da expressão> será exibido na coluna *Valor*. Os campos de monitoração em linha na parte de implementação exibirão três pontos de interrogação cada.

Expressão	Tipo	Valor	Valor Preparado
in	INT	<Valor da express...>	
out	INT	<Valor da express...>	
ivar	INT	<Valor da express...>	
1	out[???] :=in[???] +ivar[??] ;		

**Figura 5-40. Visualização Online do Bloco Funcional FB1**

### Forçamento de Variáveis

Além da possibilidade de inserir um valor preparado para uma variável na declaração de qualquer editor, o editor ST permite clicar na caixa de monitoração de uma variável na parte de implementação (no modo online), onde o valor preparado pode ser inserido (Figura 5-41).



**Figura 5-41. Diálogo Preparar Valor**

Neste diálogo, encontram-se a *Expressão* (nome da variável seguido pelo caminho na árvore de dispositivos), seu *Tipo* e *Valor Atual*. Ativando o item correspondente, escolha uma das opções:

- *Preparar um novo valor para a próxima operação de escrita ou forçamento*
- *Remover preparação com um valor*
- *Liberar o forçamento sem modificar o valor*
- *Liberar o forçamento e restaurar a variável para o valor anterior a este*

Os valores selecionados serão atribuídos na execução do comando *Forçar Valores* (menu *Comunicação*), ou pressionando <F7>.

### Posições de Breakpoint no Editor ST

Basicamente, um breakpoint localiza-se nas posições de uma POU nas quais os valores das variáveis podem mudar, ou nas quais o fluxo do programa estende-se e outra POU é chamada. Nas seguintes descrições, “{BP}” indica uma possível posição de breakpoint.

- **Atribuição:** no início da linha. Considere que atribuições, como expressões, não definem outras posições de breakpoint em uma linha.

No laço FOR, está antes da inicialização do contador, antes do teste do contador e antes de uma sentença.

```
{BP} FOR i := 12 TO {BP} x {BP} BY 1 DO
{BP} [sentença 1]
...
{BP} [sentença-2]
END_FOR
```

- **Laço WHILE:** está antes do teste da condição e antes de uma sentença

```
{BP} WHILE i < 12 DO
{BP} [sentença 1]
...
{BP} [sentença-1]
END_WHILE
```

- **Laço REPEAT:** está antes do teste da condição e antes de uma sentença

```
REPEAT
```

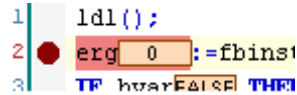
```

{BP} [sentença 1]
...
{BP} [sentença n-1]
{BP} UNTIL i >= 12
END_REPEAT

```

- Chamada de um programa ou bloco funcional: localiza-se no início da linha
- Ao final de uma POU: no passo a passo, esta posição também será alcançada após uma instrução RETURN

#### Exibição do Breakpoint em ST

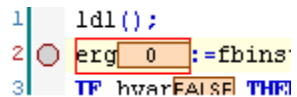


```

1 | ldl();
2 | erg 0 :=fbinst;
3 | TF hvarFAISE THEN

```

Figura 5-42. Breakpoint no Modo Online

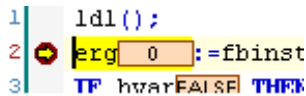


```

1 | ldl();
2 | erg 0 :=fbinst;
3 | TF hvarFAISE THEN

```

Figura 5-43. Breakpoint Desabilitado



```

1 | ldl();
2 | erg 0 :=fbinst;
3 | TF hvarFAISE THEN

```

Figura 5-44. Parada de Programa em um Breakpoint

NOTA: Um breakpoint será automaticamente definido em todos os métodos em que puder ser chamado. Se um método for chamado via um ponteiro em um bloco funcional, os breakpoints serão configurados no método do bloco funcional e em todos os blocos funcionais derivados que o subscrevem.

## Editor FBD/LD/IL

O editor FBD/LD/IL fornece comandos para trabalhar na linguagem de programação Diagrama de Blocos Funcionais (FBD), Diagrama Ladder (LD) e Lista de Instruções (IL).

### Diagrama de Blocos Funcionais - FBD

O Diagrama de Blocos Funcionais é uma linguagem de programação orientada graficamente. Ele funciona com uma lista de redes, cada qual com uma estrutura gráfica de caixas e linhas de conexão que representam tanto uma expressão lógica quanto aritmética, uma chamada de um bloco funcional, um salto ou uma instrução de retorno.

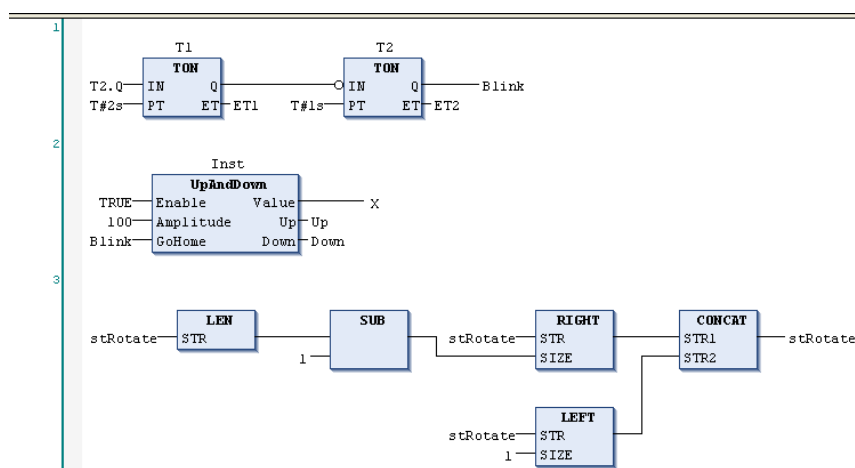


Figura 5-45. Redes de Diagrama de Blocos Funcionais

### Diagrama Ladder - LD

O Diagrama Ladder é uma linguagem de programação gráfica similar à estrutura de um circuito elétrico.

O Diagrama Ladder é adequado para construir intertravamentos lógicos, mas pode também criar redes como no FBD. Portanto, o LD é bastante útil para controlar a chamada de outras POU's.

Ele consiste de uma série de redes, sendo cada uma delas limitada a sua direita e esquerda por linhas de corrente verticais (barras de energia). Uma rede contém um diagrama de circuito formado por contatos, bobinas, opcionalmente POU's adicionais (caixas) e por linhas conectoras. No lado esquerdo há uma série de contatos que transmitem a condição ON ou OFF da esquerda para a direita (o que corresponde aos valores booleanos TRUE e FALSE). Para cada contato, é atribuída uma variável Booleana.

No caso de contatos normalmente abertos, se esta variável for TRUE, a condição será transmitida da esquerda para a direita ao longo da linha conectora. Por outro lado se esta variável for FALSE, a condição transmitida para a direita será sempre OFF.

No caso de contatos normalmente fechados, se esta variável for TRUE, a condição transmitida para a direita será sempre OFF. No entanto no caso em que a variável for FALSE, a condição será transmitida da esquerda para a direita ao longo da linha conectora.

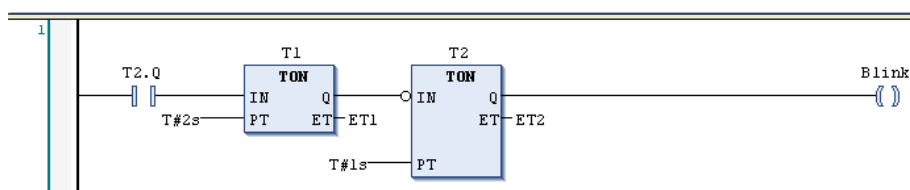


Figura 5-46. Rede LD

### Lista de Instruções - IL

A Lista de Instruções é similar a linguagem de programação Assembly, em conformidade a IEC 61131-3.

Esta linguagem suporta programações baseadas em um acumulador. Todos operadores IEC 61131-3 são permitidos, assim como as várias entradas/saídas, negações, comentários, saídas set/reset e saltos incondicionais/condicionais.

Cada instrução, fundamentalmente, é baseada no carregamento de valores no acumulador através do uso do operador LD. Em seguida, a operação é executada com o primeiro parâmetro retirado do

acumulador. O resultado da operação novamente é disponibilizado no acumulador, a partir de onde o usuário deve armazená-lo com a instrução ST.

Para programar execuções condicionais ou laços, a IL suporta tanto operadores de comparação (EQ, GT, LT, GE, LE, NE), quanto saltos. Estes últimos podem ser incondicionais (JMP) ou condicionais (JMPC/JMPCN). Para saltos condicionais, o valor do acumulador é verificado em TRUE ou FALSE.

Uma lista de instruções (IL) consiste de uma série de instruções. Cada instrução inicia em uma nova linha e contém um operador. Dependendo do tipo da operação, ela pode conter também um ou mais operandos separados por vírgulas. O operador deve ser estendido por um modificador.

Na linha anterior à instrução, pode haver uma marca de identificação (rótulo) seguida de dois pontos, por exemplo “ml:” no exemplo mostrado abaixo. Um rótulo pode ser o destino de uma instrução de salto, por exemplo “JMPC next” no exemplo mostrado abaixo.

Um comentário deve ser posicionado como o último elemento de uma linha.

Linhas vazias podem ser inseridas entre instruções.

```

1 PROGRAM IL
2 VAR
3   inst1: TON;
4   dwVar: DWORD;
5   dwRes: DWORD;
6   t1: TIME;
7   tout1: TIME;
8   inst2: TON;
9   bVar: BOOL;
10 END_VAR

```

---

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

LD	bVar			variable
ST	inst1.IN			starts timer with risin...
JMPC	ml			
CAL	inst1{			
	PT:=t1,			
	ET:=>tout1)			
LD	inst1.Q			is TRUE, PT seconds aft...
ST	inst2.IN			starts timer with risin...
ml:				
LD	dwVar			
ADD	230			
ST	dwRes			

Figura 5-47. Exemplo de Programa IL no Editor Tabular IL

O editor IL é um editor tabular integrado ao editor FBD/LD/IL.

### Modificadores e Operadores em IL

Os seguintes modificadores podem ser usados em uma Lista de Instruções:

Modificador	Contexto	Descrição
C	Com JMP, CAL, RET	A instrução somente será executada se o resultado da expressão anterior for TRUE.
N	Com JMPC, CALC, RETC	A instrução somente será executada se o resultado da expressão anterior for FALSE.
N	Em qualquer outro caso	Negação do operando (não do acumulador).

Tabela 5-11. Modificadores

A Tabela 5-12 apresenta quais operadores podem ser usados em combinação com os modificadores especificados.

O acumulador sempre armazena o valor atual resultante a partir da operação anterior.

Operador	Modificadores	Significado	Exemplo
LD	N	Carrega o valor (negado) do operando para o acumulador.	LD iVar
ST	N	Armazena o conteúdo (negado) do acumulador para o operando.	ST iErg
S		Leva o operando (tipo BOOL) para TRUE quando o conteúdo do acumulador for TRUE.	S bVar1
R		Leva o operando (tipo BOOL) para FALSE quando o conteúdo do acumulador for TRUE.	R bVar1
AND	N,(	AND bit a bit do acumulador e do operando (negado).	AND bVar2
OR	N,(	OR bit a bit do acumulador e do operando (negado).	OR xVar
XOR	N,(	OR EXCLUSIVO bit a bit do acumulador e do operando (negado).	XOR N,(bVar1,bVar2)
NOT		Negação bit a bit do conteúdo do acumulador.	
ADD	(	Soma do acumulador e operando. O resultado é copiado para o acumulador.	ADD (iVar1,iVar2)
SUB	(	Subtração do acumulador e operando. O resultado é copiado para o acumulador.	SUB iVar2
MUL	(	Multiplicação do acumulador e operando. O resultado é copiado para o acumulador.	MUL iVar2
DIV	(	Divisão do acumulador e operando. O resultado é copiado para o acumulador.	DIV 44
GT	(	Verifica se o acumulador é maior que (>) o operando. O resultado (BOOL) é copiado para o acumulador.	GT 23
GE	(	Verifica se o acumulador é maior ou igual (>=) ao operando. O resultado (BOOL) é copiado para o acumulador.	GE iVar2
EQ	(	Verifica se o acumulador é igual (=) ao operando. O resultado (BOOL) é copiado para o acumulador.	EQ iVar2
NE	(	Verifica se o acumulador é diferente (<>) do operando. O resultado (BOOL) é copiado para o acumulador.	NE iVar1
LE	(	Verifica se o acumulador é menor ou igual (<=) ao operando. O resultado (BOOL) é copiado para o acumulador.	LE 5
LT	(	Verifica se o acumulador é menor (<) que o operando. O resultado (BOOL) é copiado para o acumulador.	LT cVar1
JMP	CN	Salto incondicional para o rótulo.	JMPN next
CAL	CN	Chamada (Condicional) de PROGRAMA ou BLOCO FUNCIONAL (se o acumulador for TRUE).	CAL prog1
RET		Retorno antecipado da POU e do salto para a POU onde ocorreu a chamada.	RET
RET	C	Condicional - se o acumulador for TRUE... Retorno antecipado da POU e do salto para a POU onde ocorreu a chamada.	RETC
RET	CN	Condicional - se o acumulador for FALSE... Retorno antecipado da POU e do salto para a POU onde ocorreu a chamada.	RETCN
)		Avalia a operação de referência	

Tabela 5-12. Operadores e Modificadores

Consulte o item **Operadores**.



Para obter informações sobre o uso e tratamento de operandos múltiplos, operandos complexos, função/método/bloco funcional/programação, chamadas e saltos, veja o item **Trabalhando na Visualização do Editor IL**.

1	<b>AND</b>	TRUE	load TRUE to accumulator
	<b>ANDN</b>	bVar1	execute AND with negated value of bVar1
	<b>JMPC</b>	m1	if accum. is TRUE, jump to label "m1"
	<b>LDN</b>	bVar2	store negated value of bVar2...
	<b>ST</b>	bRes	... in bRes
2		<u>m1:</u>	
	<b>LD</b>	bVar2	store value of bVar2...
	<b>ST</b>	bRes	... in bRes

Figura 5-48. Programa IL com Uso de Operadores

### Trabalhando na Visualização dos Editores FBD e LD

Redes são as unidades básicas da programação FBD e LD. Cada rede contém uma estrutura que exhibe uma expressão lógica ou aritmética, uma POU (função, programa, chamada de bloco funcional, etc.), um salto ou uma instrução de retorno.

Ao criar um novo objeto, a janela do editor automaticamente contém uma rede vazia.

Consulte as configurações gerais do editor no diálogo *Opções*, guia *Editores FBD, LD e IL*.

Quando o cursor está posicionado sob nome de uma variável ou parâmetro de caixa, ele exhibe uma tooltip que indica o respectivo tipo e, no caso de instâncias de blocos funcionais, o valor de inicialização. Para os operadores IEC SEL, LIMIT, MUX, uma breve descrição das entradas será exibida. Se definido, também serão exibidos o endereço e comentário do símbolo, assim como o comentário do operando (entre aspas na segunda linha).

Inserindo e organizando elementos:

- Elementos também podem ser diretamente arrastados com o mouse a partir da caixa de ferramentas para a janela do editor ou de uma posição para outra. Para isto, selecione o elemento com um clique de mouse, mantenha-o pressionado e arraste o mesmo para a respectiva rede na visualização do editor (arrastar e soltar). Assim que a rede tiver sido alcançada, todas as possíveis posições de inserção serão indicadas por marcadores verdes. Ao posicionar o cursor em um destes pontos, o marcador mudará para verde e, soltando o botão, o elemento poderá ser ali posicionado.
- Os comandos *Recortar*, *Copiar*, *Colar* e *Excluir*, por padrão disponíveis no menu *Editar*, podem ser usados para organizar elementos. A cópia também pode ser feita via arrastar e soltar, selecione o elemento na rede com um clique de mouse e pressione a tecla <CTRL> do teclado, mantendo-os pressionados, arraste o elemento para a posição destino. Assim que a posição for alcançada (marcador de posição verde), um símbolo de mais será acrescentado ao símbolo do cursor. Solte o botão do mouse e insira o elemento.
- Para saber as posições possíveis do cursor, consulte **Posições do Cursor em FBD, LD e IL**.

Percorrendo a tabela:

- As teclas de seta devem ser usadas para saltar para a posição do cursor (anterior e posterior) e entre as redes.
- A tecla <TAB> deve ser usada para saltar para a posição do cursor (anterior e posterior) dentro da rede.
- As teclas <CTRL>+<HOME> rolam para o início do documento e marcam a primeira rede.
- As teclas <CTRL>+<END> rolam para o fim do documento e marcam a última rede.
- A tecla <PAGEUP> rola uma tela para cima e marca o retângulo superior (mais acima).
- A tecla <PAGEDOWN> rola uma tela para baixo e marca o retângulo superior (mais acima).

Selecioneando:

- Um elemento (rede) pode ser selecionado clicando-se na respectiva posição do cursor ou usando-se as teclas de setas ou de tabulação.
- A seleção de vários elementos não-adjacentes e redes pode ser feita mantendo estes elementos pressionados ao selecionar os elementos desejados, um após o outro.
- No editor LD, a seleção de vários elementos adjacentes ou redes pode ser feita mantendo-se simultaneamente pressionadas as teclas <SHIFT> e os dois contatos que determinam o início e o fim da seção de rede desejada, essa forma de seleção não se aplica a elementos do tipo bobina. Para recortar (copiar) e colar uma seção de uma rede, basta manter simultaneamente pressionadas as teclas <CTRL> e os dois contatos que definem as bordas desta seção. Desta forma, os elementos entre eles serão automaticamente considerados.

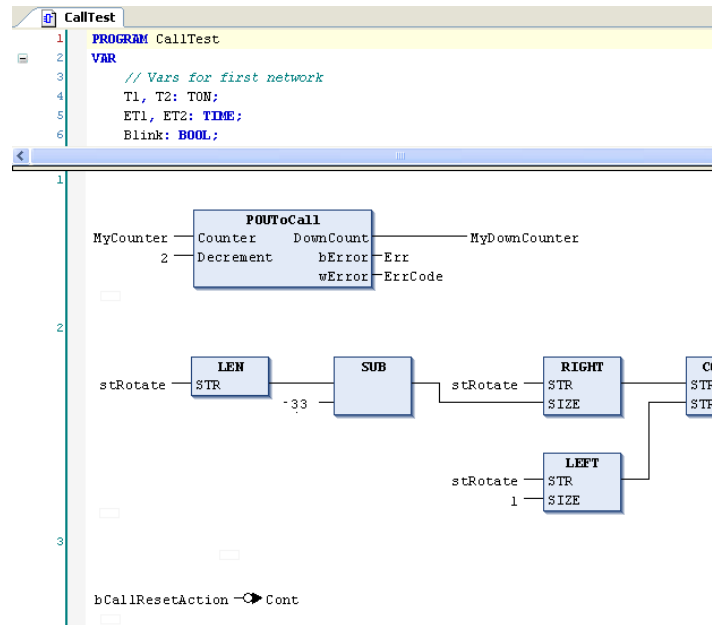


Figura 5-49. Janela do Editor FBD

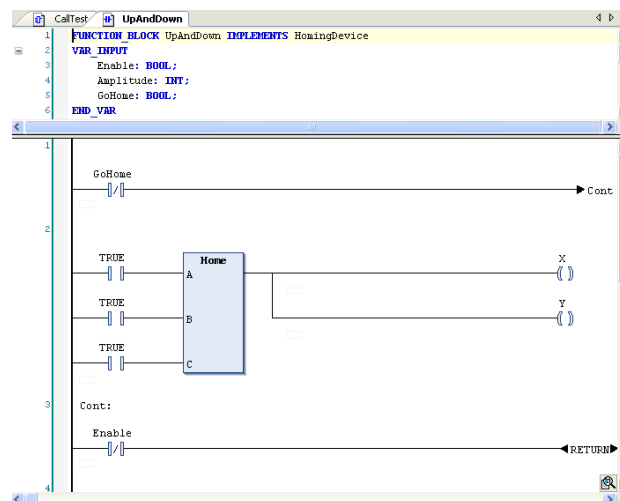


Figura 5-50. Janela do Editor LD

Para obter informações sobre as linguagens, consulte:

- **Diagrama de Blocos Funcionais - FBD**
- **Diagrama Ladder - LD**

## Trabalhando na Visualização do Editor IL

O editor IL (Lista de Instruções) é um editor tabular, diferentemente do editor textual puro usado no MasterTool IEC. A estrutura de rede dos programas FBD ou LD é também representada em um programa IL. Basicamente, uma rede é suficiente em um programa IL, porém, considerando a troca entre FBD, LD e IL, também é possível usar redes para estruturar um programa IL.

Verifique as configurações gerais do editor no diálogo *Opções*, guia *Editores FBD, LD e IL*.

Para informações sobre variáveis ou parâmetros de caixas consulte **Trabalhando na Visualização dos Editores FBD e LD**.

Inserindo e organizando elementos:

Os comandos para trabalhar neste editor estão disponíveis no menu *FBD/LD/IL*, que é sempre o mais importante no menu de contexto.

Cada unidade de programação (elemento) é inserida na posição atual do cursor através do comando *Inserir* (disponível no menu *FBD/LD/IL*).

Os comandos *Recortar*, *Copiar*, *Colar* e *Excluir* por padrão estão disponíveis no menu *Editar* e podem ser usados para organizar os elementos. Considere as posições possíveis do cursor.

Veja a seguir como o editor tabular é estruturado e como é possível navegar por ele utilizando operandos complexos, chamadas e saltos.

### Estrutura do Editor Tabular IL

Cada linha do programa é escrita em uma linha da tabela estruturada nas seguintes colunas:

Coluna	Contém	Descrição
1	Operador	Este campo contém o operador IL (LD, ST, CAL, AND, OR etc.) ou um nome de função. No caso de uma chamada de bloco funcional, são especificados os respectivos parâmetros. Neste caso, devem ser inseridos os campos Prefixo ":@" ou "=>".
2	Operando	Este campo contém exatamente um operando ou um rótulo de salto. Se for necessário mais de um operando (operadores múltiplos/extensíveis "AND, A, B, C" ou chamadas de função com vários parâmetros), estes devem ser escritos em uma das linhas seguintes onde o campo do operador deve ser deixado vazio. Neste caso, acrescente um parâmetro separado por vírgulas. No caso de um bloco funcional, programa ou chamada de ação, deve ser acrescentada uma correta abertura e/ou fechamento de parênteses.
3	Endereço	Este campo contém o endereço do operando, conforme definido na parte de declaração. O campo não pode ser editado e pode ser habilitado ou desabilitado através da opção <i>Mostrar Endereço do Símbolo</i> .
4	Comentário de símbolo	Este campo contém o comentário, conforme definido para o operando na parte de declaração. O campo não pode ser editado e pode ser habilitado ou desabilitado através da opção <i>Mostrar Endereço do Símbolo</i> .
5	Comentário de operando	Este campo contém o comentário para a linha atual. É editável e pode ser habilitado ou desabilitado através da opção <i>Mostrar Endereço do Símbolo</i> .

**Tabela 5-13. Estrutura do Editor Tabular**

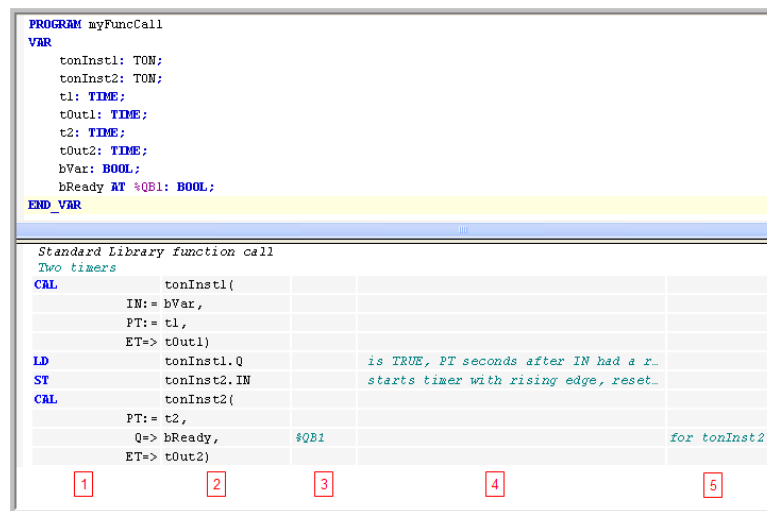


Figura 5-51. Editor Tabular IL

Percorrendo a tabela:

- Teclas <↑> e <↓>: possibilitam o deslocamento para os campos superior e inferior respectivamente.
- <TAB>: movimenta-se para o campo à direita.
- <SHIFT> + <TAB>: movimenta-se na linha para o campo à esquerda.
- <SPACE>: abre o campo atualmente selecionado. Pode ser aberto com dois cliques no mouse. Se aplicável, o Assistente de Entrada pode ser disponibilizado através do botão . O campo aberto pode ser fechado com <ENTER> ou <ESC>, confirmando ou cancelando as escolhas feitas, respectivamente.
- <CTRL> + <ENTER>: insere uma nova linha abaixo da atual.
- <DELETE>: exclui a linha atual, ou seja, onde o campo foi selecionado.
- *Recortar, Copiar, Colar*: para copiar uma ou mais linhas, selecione no mínimo uma e use o comando correspondente. Para recortar uma linha, use o comando *Recortar*. O comando *Colar* insere a linha previamente copiada/recortada antes do campo selecionado. Se não houver campo selecionado, a linha será colada ao final da rede.
- <CTRL> + <HOME>: rola para o início do documento e marcam a primeira rede.
- <CTRL> + <END> : rola para o fim do documento e marcam a última rede.
- <PAGEUP>: rola uma tela para cima e marca o retângulo superior (mais acima).
- <PAGEDOWN>: rola uma tela para baixo e marca o retângulo superior (mais acima).

### Múltiplos Operandos (Operadores Extensíveis)

Se o mesmo operador for usado com vários operandos, dois caminhos de programação são possíveis:

- Os operandos são inseridos em linhas subsequentes, separados por vírgulas.

Exemplo em IL:

4	LD	7
	ADD	2,
		4,
		7
	ST	iVar

- A instrução é repetida nas linhas subsequentes.

Exemplo em IL:

3	<b>LD</b>	7
	<b>ADD</b>	2
	<b>ADD</b>	4
	<b>ADD</b>	7
	<b>ST</b>	iVAR

### Operandos Complexos

Para usar um operando complexo, digita-se um parênteses de abertura, usa-se as linhas seguintes para os componentes do operando específico e, abaixo destes, em uma linha separada, digita-se o parênteses de fechamento.

Exemplo em IL para rotacionar uma string em 1 caractere a cada ciclo:

4	<b>LD</b>	stRotate	
	<b>RIGHT (</b>	stRotate	
	<b>LEN</b>		
	<b>SUB</b>	1	
	<b>)</b>		
	<b>CONCAT (</b>	stRotate	
	<b>LEFT</b>	1	
	<b>)</b>		
	<b>ST</b>	stRotate	

### Chamadas de Função

Digite o nome da função no campo do operador. O primeiro parâmetro de entrada deve ser fornecido como um operando em uma operação LD anterior. Se houver parâmetros adicionais, o próximo deve ser fornecido na mesma linha que o nome da função. Os demais também podem ser adicionados nesta linha, separados por vírgulas, ou nas linhas subsequentes.

O valor de retorno da função será armazenado no acumulador, mas observe a seguinte restrição em relação à norma IEC: não é possível realizar uma chamada de função com múltiplos valores de retorno. Somente um valor pode ser usado para se obter uma operação bem sucedida.

Exemplo: foi chamada a função *GeomAverage*, que apresenta três parâmetros de entrada. O primeiro parâmetro é fornecido por X7 na operação precedente. O segundo, 25, é dado antes do nome da função. O terceiro parâmetro é dado pela variável TVAR, tanto na mesma linha quanto na subsequente. O valor de retorno é atribuído à variável AVE.

Exemplo para chamada da função *GEOM AVERAGE* em ST:

```
AVE := GEOM AVERAGE (X7, 25, TVAR);
```

Exemplo para chamada da função *GEOM AVERAGE* em IL:

<b>LD</b>	X7
<b>GeomAverage</b>	25
<b>ST</b>	Ave

### Chamadas de Blocos Funcionais e Chamadas de Programa

Use os operadores *CAL* ou *CALC*. Insira o nome da instância do bloco funcional e do programa no campo do operando (segunda coluna), seguido pelo parêntese de abertura. Digite os parâmetros de entrada em cada uma das seguintes linhas:

Primeira coluna: operador (nome do parâmetro) e:

- “:=” para parâmetros de entrada
- “=>” para parâmetros de saída

Segunda coluna: operando (parâmetro atual) e:

- “,” se seguido de outros parâmetros
- “)” após o último parâmetro
- “()” no caso de chamadas sem parâmetros

Exemplo de chamada da POUToCall com dois parâmetros de entrada e dois de saída:

```

1  PROGRAM IL_EXAMPLE
2  VAR
3      bErr: BOOL;
4      wResult: WORD;
5  END_VAR
6

```

---

```

1
   CAL          POUToCall (
      iCounter:= 1,
      iDecrement:= 1000,
      wError=> wResult)
   LD          POUToCall.bError
   ST          bErr

```

Não é necessário usar todos parâmetros de um bloco funcional ou programa.

NOTA: Como uma restrição da norma IEC, expressões complexas não podem ser usadas. Elas devem ser atribuídas à entrada do bloco funcional ou programa antes da chamada de instrução.

### Chamada de Ação

Deve ser realizada conforme um bloco funcional ou chamada de programa. O nome da ação deve ser anexado ao nome da instância ou do programa.

Exemplo de chamada da ação ResetAction em IL:

```

CAL          Inst.ResetAction()

```

### Chamada de Método

Deve ser realizada conforme uma chamada de função. O nome da instância com o nome do método anexado deve ser digitado na primeira coluna (operador).

Exemplo de chamada do método Home em IL:

```

LD          TRUE
IHome.Home TRUE ,
           TRUE
ST          Z

```

### Salto

Um salto é programado com JMP na primeira coluna (operador) e com o nome do rótulo na segunda (operando). O rótulo deve ser definido na rede destino, no campo rótulo. Observe que a lista de declaração anterior ao salto incondicional deve ser encerrada com um dos seguintes comandos: ST, STN, S, R, CAL, RET ou outra JMP. O mesmo não ocorre com o salto condicional: este é programado por JMPC na primeira coluna (operador). A execução do salto depende do valor

carregado. Exemplo em IL de uma instrução de salto condicional, se bCallResetAction for TRUE, o programa deve pular para a rede rotulada com Cont:

<b>LDN</b>	bCallResetAction
<b>JMPC</b>	Cont

## Posições do Cursor em FBD, LD e IL

### Editor IL

Este é um editor textual, estruturado sob a forma de tabela. Cada célula é uma posição possível do cursor. Consulte: **Trabalhando na Visualização do Editor IL**.

### Editores FBD e LD

Estes são editores gráficos. Os seguintes exemplos mostram as posições possíveis do cursor: texto, entrada, saída, caixa, contato, bobina, linha entre elementos, subrede e rede. Ações como recortar, copiar, colar, excluir e outros comandos específicos do editor referem-se à posição atual do cursor e ao elemento selecionado. Consulte: **Trabalhando na Visualização dos Editores FBD e LD**.

No FBD, basicamente, há um retângulo pontilhado em volta do respectivo elemento indicando a posição atual do cursor e, adicionalmente, os textos e caixas adquirem um sombreado azul e vermelho respectivamente.

Nas bobinas LD os contatos tornam-se vermelhos assim que o cursor é nelas posicionado.

A posição do cursor determina quais elementos estão disponíveis no menu de contexto para serem inseridos.

Posições possíveis do cursor:

Nos campos de texto: na figura da esquerda, as posições possíveis do cursor estão marcadas com uma moldura vermelha. A figura da direita mostra uma caixa com o cursor posicionado no campo AND. Observe a possibilidade de digitar endereços em vez de nomes de variáveis (desde que configurados adequadamente no diálogo *Opções*, guia *Editores FBD, LD e IL*).

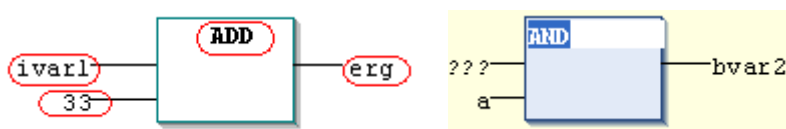


Figura 5-52. Campos de Texto

- Em cada entrada:



Figura 5-53. Entradas

- Em cada operador, função ou bloco funcional:



Figura 5-54. Operador, Função ou Bloco Funcional

- Saídas, se uma atribuição ou um salto vier logo em seguida:



Figura 5-55. Saída

- Imediatamente antes do pontilhado na atribuição, salto ou instrução de retorno:

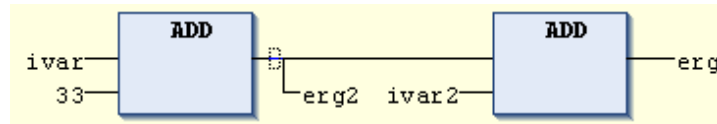


Figura 5-56. Antes do Pontilhado

- A posição do cursor mais a direita na rede ou em qualquer outro lugar desta além das outras posições possíveis. Isto selecionará toda a rede:

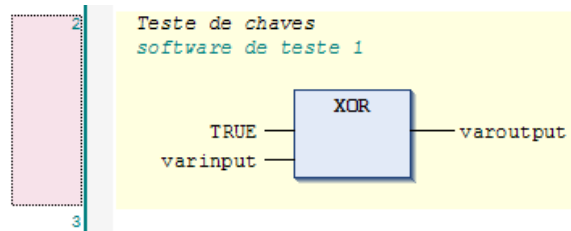


Figura 5-57. Posição do Cursor a Direita na Rede

- O pontilhado diretamente a frente de uma atribuição:



Figura 5-58. Posição a Frente da Atribuição

- Em cada contato:

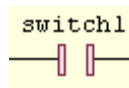


Figura 5-59. Contato

- Em cada bobina:

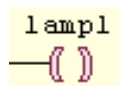


Figura 5-60. Bobina



- A linha de conexão entre contatos e bobinas:

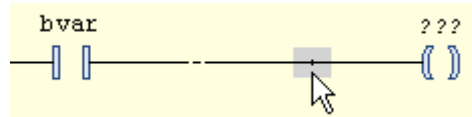


Figura 5-61. Posição na Linha de Conexão

- Em uma ramificação ou subrede:

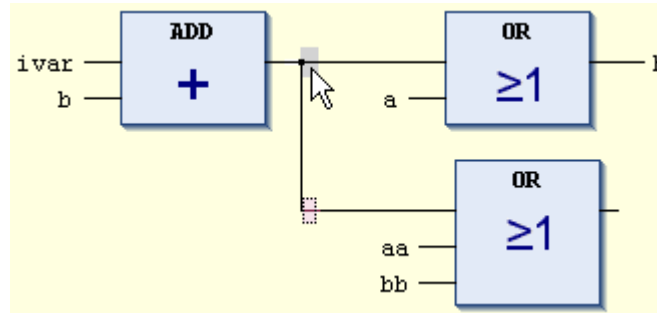


Figura 5-62. Ramificação ou Subrede

### Menu FBD/LD/IL

Quando o cursor está localizado na janela do editor FBD/LD/IL, o menu deste está disponível na barra do menu e fornece os comandos para programação na visualização do editor atualmente configurado.

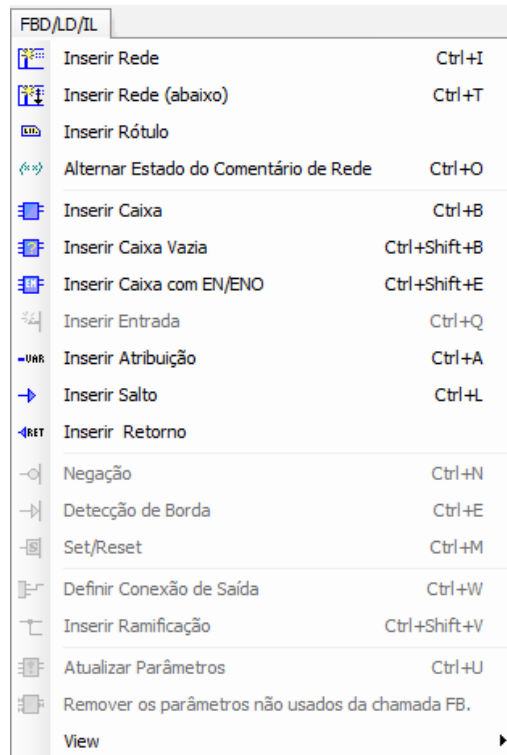


Figura 5-63. Menu FBD/LD/IL na Visualização do Editor FBD

Para obter uma descrição dos comandos, consulte **Comandos do Editor FBD/LD/IL** no Manual de Utilização MasterTool IEC XE – MU299048.

## Elementos

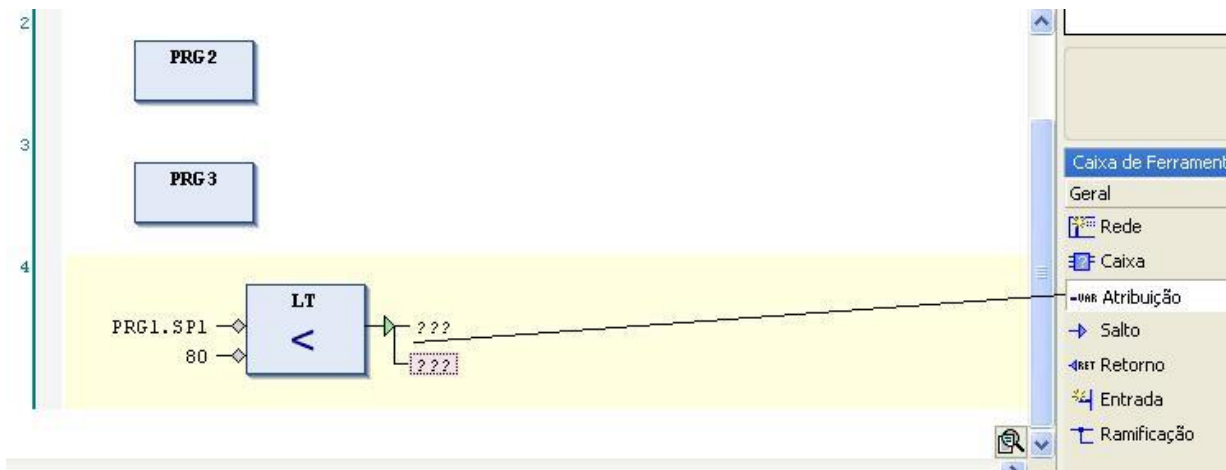
### Caixa de Ferramentas FBD/LD/IL

O editor FBD/LD/IL fornece uma caixa de ferramentas com elementos de programação a serem inseridos na janela do editor (arrastar e soltar). Por padrão, esta caixa de ferramentas pode ser aberta via comando *Caixa de Ferramentas* no menu *Visualizar*.

Os elementos que estarão disponíveis para inserção dependem da visualização do editor atualmente ativo (veja a respectiva descrição dos comandos Inserir). Os elementos são classificados em categorias: *Geral* (elementos gerais como Rede, Atribuição, etc.), *Operadores Booleanos*, *Operadores Matemáticos*, *Outros Operadores* (SEL, MUX, LIMIT e MOVE), *Blocos Funcionais* (R\_TRIG, F\_TRIG, RS, SR, TON, TOF, CTD e CTU), *Elementos Ladder* e *POUs* (definidas pelo usuário).

A categoria POU's lista todas as POU's definidas pelo usuário, como o objeto FBD/LD/IL atualmente aberto no editor. A lista será atualizada automaticamente quando POU's forem adicionadas ou removidas da aplicação.

As pastas da categoria podem ser abertas através de um clique na respectiva categoria. Veja na Figura 5-64: a categoria *Geral* está aberta, e as outras estão fechadas, ela mostra um exemplo para inserção de uma atribuição (através de arrastar e soltar a partir da caixa de ferramentas). Neste caso, somente está aberta na caixa de ferramentas a seção *Geral*.



**Figura 5-64. Inserção A Partir da Caixa de Ferramentas**

Para inserir um elemento no editor, selecione-o na caixa de ferramentas com um clique de mouse e, arrastando e soltando, traga-o para a janela do editor. As posições possíveis de inserção são indicadas por marcadores de posição, que aparecem enquanto o elemento está sendo desenhado na janela do editor (mantendo-se o mouse pressionado). A posição mais próxima possível se tornará verde. Ao soltar o botão do mouse, o elemento será inserido nesta posição (verde).

Se um elemento caixa foi desenhado em uma caixa já existente, a nova substituirá a antiga; se entradas e saídas já tiverem sido atribuídas, estas serão mantidas conforme definido.

### Rede

A rede é a unidade básica de um programa FBD ou LD. Nestes editores, as redes são organizadas em uma lista vertical. Cada rede apresenta, em seu lado esquerdo, um número de rede serial e sua estrutura consiste tanto de uma expressão lógica ou aritmética quanto de um programa, função, chamada de bloco funcional, salto ou instrução de retorno.

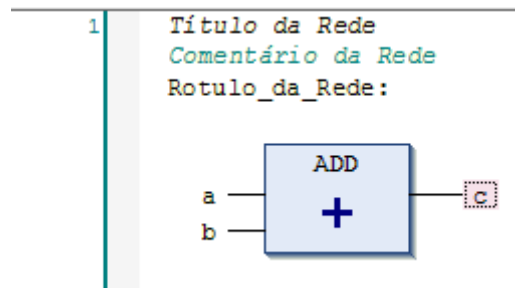
O editor IL, devido à base comum com FBD e LD, também usa o elemento rede. Se um objeto foi inicialmente programado em FBD ou LD e depois convertido para IL, as redes ainda estarão presentes no programa IL, e vice-versa. Se o usuário iniciar a programação de um objeto em IL, ele

precisará, no mínimo, de 1 elemento rede que contenha todas instruções (embora também seja possível usar outras redes para estruturar um programa, por exemplo, se considerar uma conversão para FBD ou LD).

A uma rede, podem ser opcionalmente atribuídos um título, um comentário e um rótulo:

A disponibilidade dos campos título e comentário pode ser ativada ou desativada no diálogo *Opções*, guia *Editores FBD, LD e IL*. Se a opção estiver ativada, o campo de edição do título pode ser aberto com um clique de mouse na rede, diretamente abaixo da borda superior. Para inserir um comentário, abra o campo de edição correspondente abaixo do título. O comentário pode apresentar várias linhas. Quebras de linha são inseridas via <ENTER> e a entrada do texto do comentário é encerrada com <CTRL>+<ENTER>.

Para acrescentar um rótulo, o qual pode ser endereçado por um salto, usa-se o comando *Inserir Rótulo*. Os rótulos são exibidos abaixo dos campos título e comentário e, caso estes não estejam disponíveis, diretamente abaixo da borda superior de uma rede.



**Figura 5-65. Posições do Título, Comentário e Rótulo em uma Rede**

As redes podem ser configuradas em estado de comentário, o que faz com que a rede não seja processada, mas exibida e tratada como um comentário.

Os comandos padrão para *Copiar*, *Recortar*, *Inserir* e *Excluir* podem ser aplicados a uma rede selecionada.

NOTA: Quando o clique do mouse é executado em um título, comentário ou rótulo, somente estes serão selecionados (não a rede inteira). Assim sendo, a execução dos comandos padrão não tem influência na rede em si.

Para inserir uma rede, usa-se o comando *Inserir Rede* ou arrasta-se a mesma a partir da caixa de ferramentas. Toda rede (incluindo seus elementos) pode ser copiada ou movida ao ser arrastada e solta no editor.

Considere a possibilidade de criar subredes através da inserção de ramificações.


## Rede RET

No modo online, automaticamente uma rede vazia adicional é exibida abaixo das redes existentes e é identificada por RET (em vez do número da rede). Ela representa a posição na qual a execução retornará à POU de chamada e fornecerá uma possível posição de parada.

## Atribuição em FBD/LD/IL

Dependendo da posição escolhida no FBD ou LD, uma atribuição será inserida na entrada selecionada, na saída selecionada ou ao final da rede. Em uma rede LD, uma atribuição será exibida como uma bobina.

Alternativamente, é possível arrastar o elemento a partir da *Caixa de Ferramentas* ou copiá-lo/movê-lo arrastando-o e soltando-o na visualização do editor.

Após a inserção, a string de texto “???” pode ser substituída pelo nome da variável a ser atribuída. Para isto, pode ser usado o *Assistente de Entrada* (através do botão ).

Em IL, uma atribuição é programada via instruções LD e ST. Neste contexto, veja também: **Modificadores e Operadores em IL** neste capítulo.

### Salto

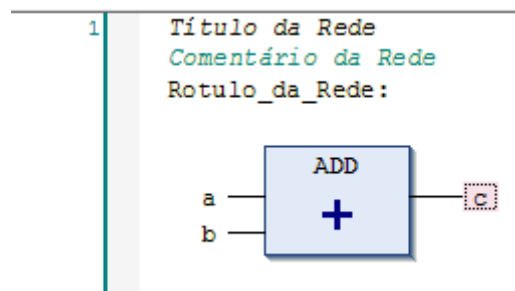
Dependendo da posição selecionada no FBD ou LD, o salto será inserido diretamente à frente da entrada selecionada, diretamente depois de saída selecionada ou ao final da rede. Alternativamente, o salto pode ser arrastado a partir da *Caixa de Ferramentas* ou copiado/movido e solto no editor.

Após a inserção, a string “???” automaticamente inserida pode ser substituída pelo rótulo ao qual o salto deve ser atribuído.

Na IL, o salto é inserido via instrução JMP.

### Rótulo

Cada FBD/LD ou rede IL apresenta, abaixo do campo de comentário de rede, um campo de entrada de texto para definir um rótulo (comando *Inserir Rótulo*). O rótulo é um identificador opcional para a rede e pode ser endereçado ao se definir o salto. Ele pode consistir de qualquer sequência de caracteres.



**Figura 5-66. Posição do Rótulo em uma Rede**

Consulte o diálogo *Opções*, guia *Editores FBD, LD e IL* para definir a exibição do comentário e o título.

### Caixas em FBD/LD/IL

Uma caixa em uma rede FBD, LD ou IL é um elemento complexo e pode representar funções adicionais, como por exemplo, Temporizadores, Contadores, Operações Aritméticas ou também programas, funções e blocos funcionais IEC.

Uma caixa pode ter entradas e saídas e ser fornecida por uma biblioteca de sistema ou programada pelo usuário. Entretanto, no mínimo uma entrada e uma saída devem ser atribuídas a valores booleanos.

Se for fornecida com o respectivo módulo, e se a opção *Mostrar Ícone da Caixa* estiver ativada, um ícone será exibido na mesma.

### Uso em FBD, LD

Uma caixa pode ser posicionada na parte esquerda de uma rede LD (como um contato) e em uma rede FBD através dos comandos *Inserir Caixa* e *Inserir Caixa Vazia*. Alternativamente, ela pode ser inserida a partir da *Caixa de Ferramentas* ou copiada/movida, arrastando-a e soltando-a na visualização do editor. Para obter mais detalhes, consulte o item **Inserir Caixa** no Manual de Utilização MasterTool IEC XE – MU299048.

## Uso em IL

Em um programa IL, será inserida uma instrução CAL com parâmetros para representar um elemento caixa.

Na implementação atual, os parâmetros da caixa (entradas, saídas) podem ser atualizados sem que a caixa tenha que ser reinserida através do comando *Atualizar Parâmetros* (no caso da interface da caixa ter sido alterada).

## Instrução RETURN em FBD/LD/IL

Com uma instrução RETURN, a execução da POU FBD, LD ou IL pode ser desviada.

Nas redes FBD ou LD, a instrução pode ser posicionada em paralelo ou ao final dos elementos anteriores. Se a entrada de RETURN for TRUE, o processamento da POU será imediatamente abortado.

Para inserir o retorno, usa-se o comando *Inserir Retorno*. Alternativamente, é possível arrastar o elemento a partir da *Caixa de Ferramentas*, copiar ou mover o mesmo a partir de outra posição no editor.

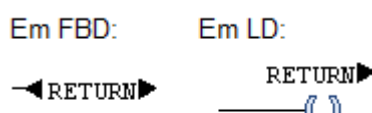


Figura 5-67. Elemento RETURN

Em IL, é usada a instrução RET.

## Ramificação/Bobina Suspensa em FBD/LD/IL

## FBD, LD

Nas redes FBD e LD, uma ramificação e uma bobina suspensa respectivamente dividem a linha de processamento a partir da posição atual do cursor. Esta linha seguirá em duas subredes, as quais são executadas uma após a outra, de cima para baixo. Cada subrede adquire uma ramificação adicional, permitindo, assim, múltiplas ramificações em uma rede.

Cada subrede adquire um marcador próprio (simbolizado por um pequeno retângulo) que pode ser selecionado para executar ações neste braço da ramificação.

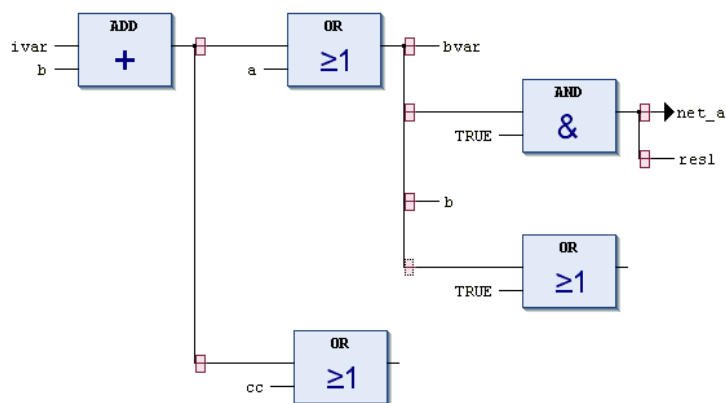


Figura 5-68. Marcadores e Ramificações

Na rede FBD, uma ramificação é inserida via comando *Inserir Ramificação*. Outra forma possível é arrastá-la a partir da *Caixa de Ferramentas*. Para verificar as posições passíveis de inserção, consulte **Inserir Ramificação** no Manual de Utilização MasterTool IEC XE – MU299048.

NOTA: *Recortar* e *Colar* não pode ser realizado em subredes.

Veja o exemplo mostrado na Figura 5-69: uma ramificação foi inserida na saída da caixa SUB. Assim, foram criadas duas subredes, cada qual seleccionável pelo seu próprio marcador. Em seguida uma caixa ADD foi adicionada em cada subrede.

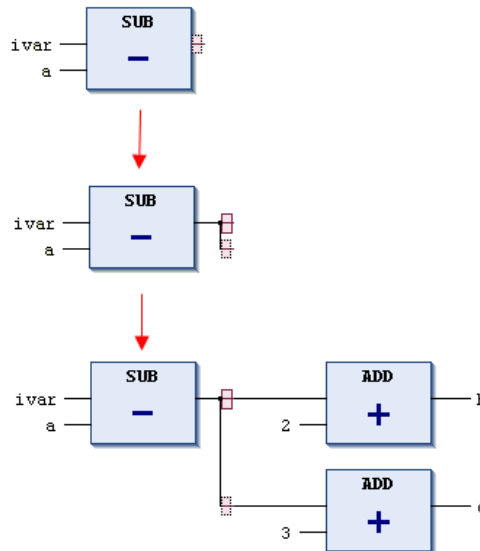


Figura 5-69. Inserção de Ramificação em FBD

Para excluir uma subrede, primeiramente remova todos os seus elementos (todos aqueles posicionados à direita do marcador da subrede). A seguir, selecione o marcador e use o comando *Excluir* (<DEL>). Veja na Figura 5-70: o elemento OR de 3 entradas deve ser excluído antes de selecionar e excluir o marcador da subrede inferior.

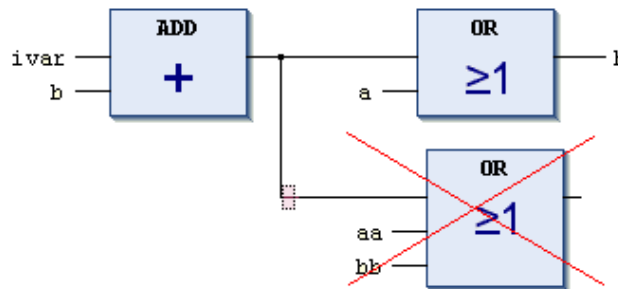


Figura 5-70. Excluir Ramificação e Subrede

Execução no modo online:

As ramificações específicas serão executadas da esquerda para a direita e de cima para baixo.

## IL (Lista de Instruções)

Em uma rede IL, a ramificação e a bobina suspensa são representadas por uma ordem apropriada de instruções. Neste contexto, consulte **Modificadores e Operadores em IL**.

## Contato

Este é um elemento LD.

Cada rede LD contém um ou mais contatos na esquerda. Estes contatos são representados por duas linhas paralelas:

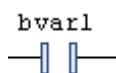


Figura 5-71. Contato

Contatos passam da condição ON (TRUE) ou OFF (FALSE) da esquerda para a direita.

Uma variável booleana é atribuída a cada contato. Se esta variável for TRUE, primeiramente a condição é transmitida da esquerda para a direita, e a seguir para uma bobina no lado direito da rede. Caso contrário, a conexão da direita recebe o valor FALSE.

Múltiplos contatos podem ser conectados tanto em série quanto em paralelo. No caso de dois contatos paralelos, pelo menos um deles deve transmitir o valor TRUE para que a ramificação paralela transmita o valor TRUE. No caso de contatos conectados em série, todos os contatos devem transmitir a condição TRUE para que o último contato transmita a condição TRUE.

Assim, o comportamento e posicionamento dos contatos é análogo aos circuitos elétricos em série e paralelo.

Um contato também pode ser negado (barra no símbolo do contato):

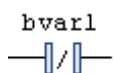


Figura 5-72. Contato Negado

Um contato negado transmite a condição de entrada (TRUE ou FALSE) somente se a variável booleana atribuída for FALSE. Observe que a caixa de ferramentas fornece diretamente elementos de contatos negados.

Um contato pode ser inserido em uma rede LD através dos comandos *Inserir Contato*, *Inserir Contato (À Direita)*, *Inserir Contato Paralelo (Acima)* ou *Inserir Contato Paralelo (Abaixo)*, os quais, por padrão, fazem parte do menu *FBD/LD/IL*. Alternativamente, o elemento pode ser inserido via arrastar e soltar a partir da *Caixa de Ferramentas* ou a partir de outra posição no editor.

## FBD, IL

Se o usuário estiver trabalhando na visualização das redes FBD ou IL, o comando não estará disponível, porém contatos e bobinas de uma rede LD podem ser representados pelos elementos FBD e pelas instruções IL correspondentes.

## Bobina

Este é um elemento LD.

Do lado direito de uma rede LD, pode haver um número indeterminado de bobinas, as quais são representadas por parênteses:



Figura 5-73. Bobina

Elas podem ser organizadas somente em paralelo. Uma bobina transmite o valor das conexões da esquerda para a direita e copia os mesmos para uma variável Booleana apropriada. Na linha de entrada, podem estar presentes os valores ON (TRUE) ou OFF (FALSE). Contatos e bobinas também podem ser negados (barra no símbolo da bobina):



**Figura 5-74. Bobina Negada**

Neste caso, o valor negado do sinal de entrada será copiado para a variável booleana apropriada. Desta forma, um contato negado somente será conectado se o valor booleano apropriado for FALSE.

Uma bobina pode ser inserida em uma rede através do comando *Inserir Atribuição*, o qual, por padrão, faz parte do menu FBD/LD. Alternativamente, o elemento pode ser inserido via arrastar e soltar a partir da *Caixa de Ferramentas* (elementos Ladder) ou a partir de outra posição no editor. Veja também: **Bobinas Set/Reset**.

#### FBD, IL

Se o usuário estiver trabalhando na visualização das redes FBD ou IL o comando não estará disponível, porém contatos e bobinas de uma rede LD podem ser representados pelos elementos FBD e instruções IL correspondentes.

#### Set/Reset em FBD/LD/IL

#### FBD, LD

Uma saída booleana em FBD, ou correspondentemente, uma bobina LD, pode ser definida como Set ou Reset. Para alternar estes estados, usam-se os comandos Set/Reset, respectivamente, a partir do menu de contexto quando a saída estiver selecionada. A saída e a bobina serão marcadas com S ou R.

**Set:** se o valor TRUE é configurado em uma saída ou bobina Set, estas se tornarão TRUE e assim permanecerão. Este valor não pode ser sobrescrito nesta posição enquanto a aplicação estiver sendo executada.

**Reset:** se o valor TRUE é configurado em uma saída ou bobina Reset, estas se tornarão FALSE e assim permanecerão. Este valor não pode ser sobrescrito nesta posição enquanto a aplicação estiver sendo executada.



**Figura 5-75. Saída Set em FBD**

Veja também: **Bobinas Set/Reset**.

#### IL

Em IL, os operadores S e R são usados para submeter um operando a set ou reset.

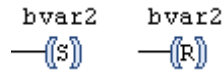
#### Bobinas Set/Reset

Através de símbolos, as bobinas podem ser definidas como do tipo Set ou Reset. A bobina Set é indicada por (S) e não permite sobrescrever o valor TRUE na variável booleana apropriada. Isto significa que a variável, uma vez definida como TRUE, permanece TRUE.

A bobina Reset é indicada pelo símbolo (R) e não permite sobrescrever o valor FALSE na variável booleana apropriada. Isto significa que a variável, uma vez definida como FALSE, permanece FALSE.

No editor LD, bobinas Set e Reset podem ser diretamente inseridas via arrastar e soltar a partir da *Caixa de Ferramentas*, categoria *Elementos Ladder*.





**Figura 5-76. Bobinas Set e Reset**

### Editores FBD/LD/IL no Modo Online

No modo online, o editor FBD/LD/IL fornece visualizações para a monitoração, escrita e forçamento das variáveis e expressões no controlador. A funcionalidade de depuração (breakpoints, passo a passo etc.) também está disponível.

Para informações sobre como abrir objetos no modo online, consulte **Interface do Usuário no Modo Online** no Manual de Utilização MasterTool IEC XE – MU299048.

Observe que a janela do editor de um objeto FBD, LD ou IL também inclui o *Editor de Declaração* na parte superior. Neste contexto, veja também: **Editor de Declaração no Modo Online** no Manual de Utilização MasterTool IEC XE – MU299048.

### Monitoração

Se a monitoração em linha não estiver explicitamente desativada no diálogo *Opções*, ela será complementada nos editores FBD ou LD por uma pequena janela de monitoração em cada variável, e também por uma coluna de monitoração adicional mostrando os valores atuais (monitoração em linha). Este ainda é o caso para entradas e saídas de blocos funcionais não atribuídos.

A janela de monitoração de uma variável mostra um pequeno triângulo vermelho no canto superior esquerdo. Se a variável atual estiver forçada, preparada para escrita ou forçamento, será indicado um triângulo azul na parte inferior esquerda. Na Figura 5-77 um exemplo de uma variável atualmente forçada e preparada para liberar o forçamento.



**Figura 5-77. Variável Forçada e Preparada para Liberar o Forçamento**

Expressão	Comentário	Tipo	Valor	Valor Preparado
Inst2		UpAndDown		
Enable		BOOL	TRUE	
Amplitude		INT	200	
GoHome		BOOL	FALSE	
Value		INT	41	
Up		BOOL	FALSE	
Down		BOOL	TRUE	
Counter		DINT	2159	
diValue		DINT	41	
X		BOOL	FALSE	
X2		INT	0	
Up		BOOL	TRUE	
Down		BOOL	FALSE	
MyCounter	Vars for third network	DINT	40	<Unforce and restore>
MyDownCounter		DINT	-2	
Err		BOOL	FALSE	
ErrCode		WORD	0	
stRotate	Vars for fifth network	STRING	'pcom'	'abc'
Ave	Vars for fifth network	DINT	38	

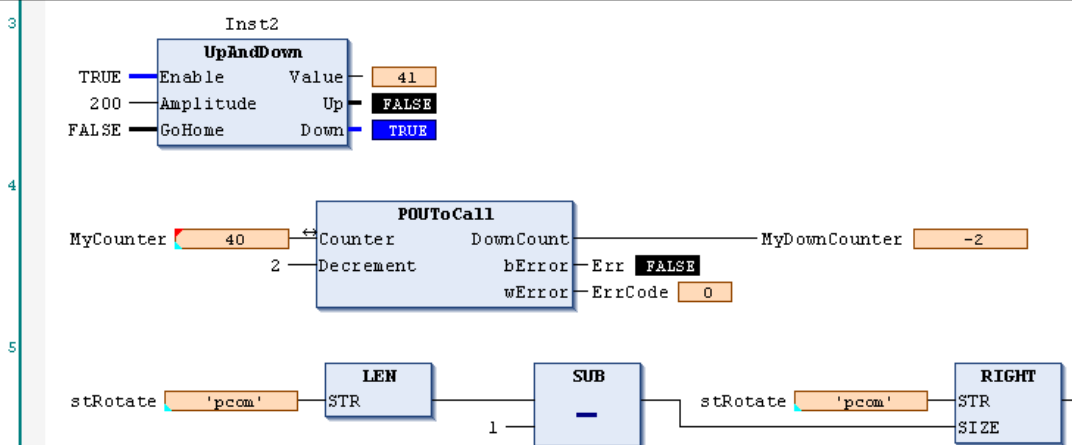


Figura 5-78. Visualização Online de um Programa FBD

iVar			
iRes			
iTemp			

1	<b>LD</b>	PLC_PRG.iX	11467	...
	<b>ADD</b>	22		add 22
	<b>ST</b>	iTemp	11488	store to temp
2	<b>LD</b>	PLC_PRG.iX	11467	
	<b>ADD</b>	1		
	<b>ST</b>	iVar	11467	
3	<b>CALL</b>	PRG1 {		call program PRG1
		iIn:= iVar,	11467	
		iOut=> iRes)	11468	store iOut to iRes

Figura 5-79. Visualização Online de um Programa IL

Na visualização online, as redes Ladder apresentam conexões animadas: conexões com valor TRUE são exibidas em negrito azul e as de valor FALSE em negrito preto. As conexões com valor desconhecido ou com valor analógico são exibidas em preto padrão.

NOTA: Os valores das conexões são calculados a partir dos valores de monitoração. Não é um fluxo de energia real.

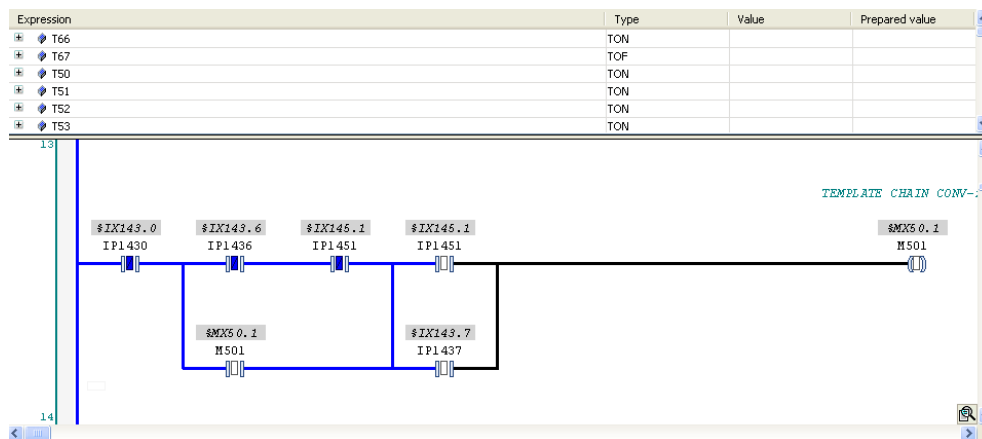


Figura 5-80. Visualização Online de um Programa LD

Observe a visualização online de uma POU do tipo bloco funcional: na parte de implementação, não serão visualizados valores nas janelas de monitoração, e sim o <Valor da expressão>. Os campos de monitoração em linha na parte de implementação apresentarão três pontos de interrogação em cada um.

### Forçamento/Escrita de Variáveis

No modo online, é possível preparar um valor para forçamento ou escrita de uma variável tanto no editor de declaração quanto na parte de implementação. Na parte de implementação, com um clique de mouse na variável, o seguinte diálogo se abrirá:

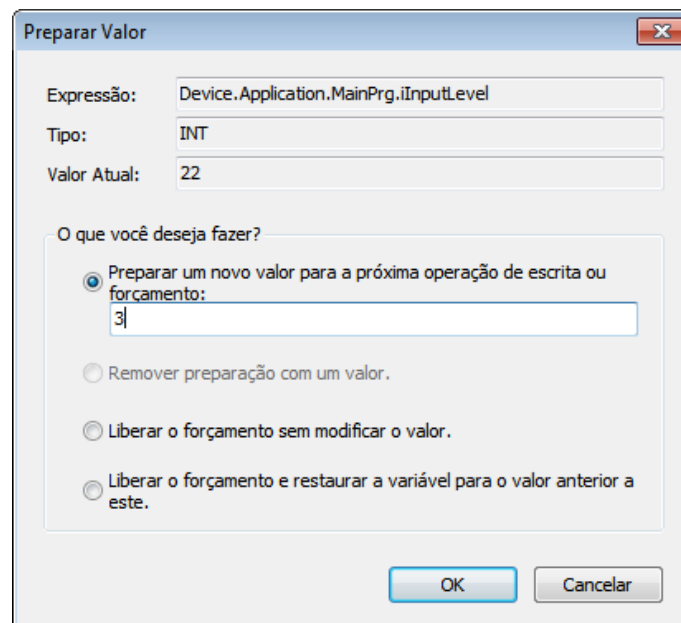


Figura 5-81. Diálogo Preparar Valor

Neste diálogo, encontra-se o nome da variável complementado pelo seu caminho na árvore de dispositivos (*Expressão*), seu *Tipo* e seu *Valor Atual*. Ativando o item correspondente, o usuário escolhe o que deseja:

- Preparar um novo valor para a próxima operação de escrita ou forçamento
- Remover a preparação com um valor

- *Liberar o forçamento sem modificar o valor*
- *Liberar o forçamento e restaurar a variável para o valor anterior a este*

A ação selecionada será transmitida após a execução do comando *Forçar Valores*, por padrão, no menu *Comunicação*, ou através da tecla <F7>.

### Breakpoint e Posições de Parada

As possíveis posições que podem ser definidas para um breakpoint (posições de parada) para fins de depuração, basicamente, são aquelas onde o valor das variáveis podem mudar (declarações), aquelas onde o fluxo do programa se ramifica, ou aquelas onde outra POU é chamada. As posições são:

- Em toda a rede que faz com que o breakpoint seja aplicado à primeira posição possível dentro da rede.
- Em uma caixa com declaração. Não é possível, portanto, em caixas como ADD e DIV. Veja a nota abaixo.
- Em uma atribuição.
- Ao final de uma POU, no ponto de retorno ao elemento de chamada. No modo online, automaticamente será exibida para este fim uma rede vazia, a qual, em vez de um número de rede, é identificada por RET.

NOTA: Não é possível configurar um breakpoint diretamente na primeira caixa de uma rede. Entretanto, se um breakpoint estiver configurado na rede inteira, a posição de parada será automaticamente aplicada à primeira caixa.

Consulte a lista de seleção no diálogo de breakpoint para verificar todas as posições possíveis.

Uma rede contendo qualquer posição de breakpoint ativa é marcada com um símbolo de breakpoint (círculo vermelho) à direita do número da rede e por um fundo com um retângulo sombreado em vermelho (para a primeira posição de breakpoint possível dentro da rede). Posições de breakpoint desativadas são indicadas por um círculo vermelho não preenchido (Figura 5-83).

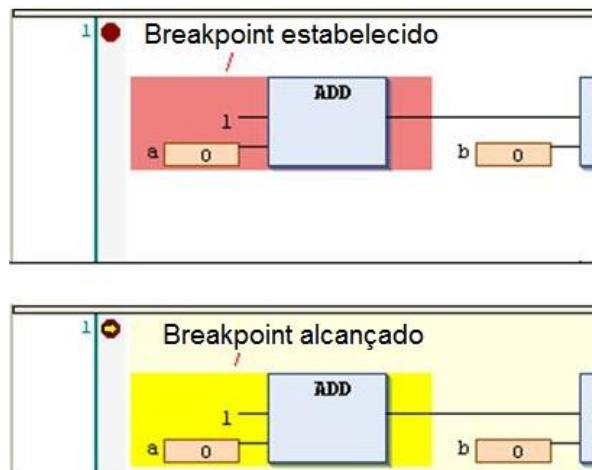


Figura 5-82. Breakpoint Configurado e Alcançado

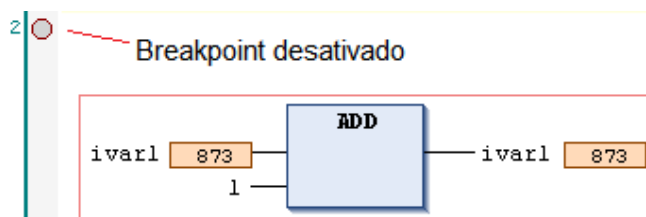


Figura 5-83. Breakpoint Desativado

Assim que uma posição de breakpoint for alcançada durante o passo a passo ou processamento do programa, uma seta amarela será exibida no símbolo do breakpoint e uma área sombreada em vermelho se tornará amarela.

A posição de parada alcançada é indicada por um sombreado amarelo e a posição não alcançada é indicada por um sombreado vermelho:

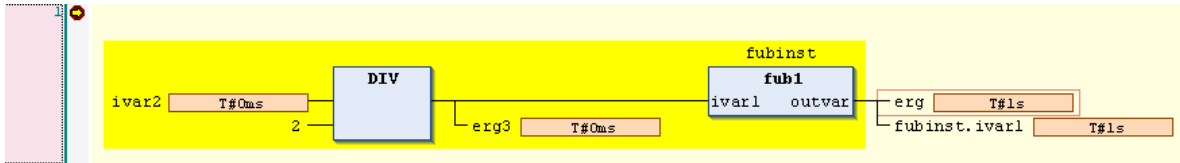


Figura 5-84. Posição de Parada Exibidas em FBD

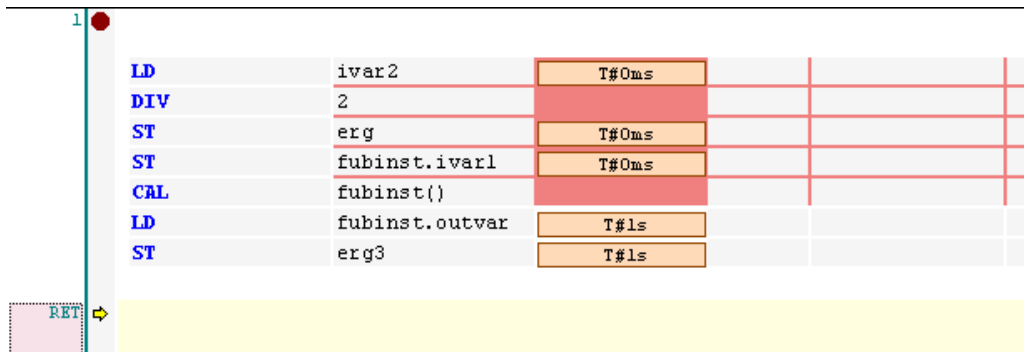


Figura 5-85. Posições de Parada Exibidas em IL

NOTA: Um breakpoint será configurado automaticamente em todos os métodos em que puder ser chamado. Se um método for chamado via ponteiro em um bloco funcional, os breakpoints serão configurados no método do bloco funcional e em todos os blocos funcionais derivados que estejam subscrevendo-o.

## 6. Bibliotecas

A biblioteca `standard.library` já vem instalada por padrão. Ela contém todas as funções e blocos funcionais necessários em conformidade com a norma IEC61131-3, assim como com as POU's para o programador IEC.

Algumas bibliotecas adicionais são necessárias, dependendo das várias funcionalidades do programador. Estas bibliotecas são usadas implicitamente e, por padrão, são incluídas automaticamente no projeto. O usuário não precisa trabalhar com elas explicitamente.

No Manual de Utilização MasterTool IEC XE – MU299048, veja também **Instalar Bibliotecas e Editor Library Manager**.

### A Biblioteca `Standard.library`

A biblioteca `standard.library` é, por padrão, fornecida com o programador MasterTool IEC XE.

Esta biblioteca contém todas as funções e blocos funcionais solicitados que correspondem à norma IEC 61131-3, tais como POU's padrão para um programador IEC. A diferença entre uma função padrão e um operador é que este é implicitamente reconhecido pelo programador enquanto aqueles devem ser vinculados ao projeto (`standard.library`).

### Funções de String

#### *LEN*

Fornecido pela `standard.library`.

Função do tipo `STRING`, retorna o comprimento de uma string.

Entrada: `STR`: `STRING`, string a ser analisada.

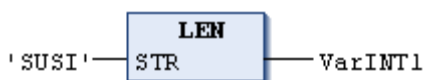
Valor de retorno: `INT`, comprimento da string (número de caracteres).

Exemplo em IL:

<b>LD</b>		'SUSI'	
<b>LEN</b>			
<b>ST</b>		VarINT1	

O resultado é "4".

Exemplo em FBD:



Exemplo em ST:

```
VarINT1 := LEN ('SUSI');
```

#### *LEFT*

Fornecido pela `standard.library`.

Função do tipo `STRING`. Retorna a string inicial da esquerda para uma determinada string.

Entradas:

`STR`: `STRING`; string a ser analisada.

`SIZE`: `INT`; comprimento da string inicial da esquerda (número de caracteres).

Valor de retorno: `STRING`; string inicial.

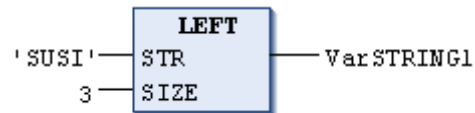
**LEFT (STR, SIZE):** retorna a quantidade de caracteres definida por SIZE, a partir da esquerda, na string STR.

Exemplo em IL:

<b>LD</b>		'SUSI'	
<b>LEFT</b>		3	
<b>ST</b>		VarSTRING1	

O resultado é “SUS”.

Exemplo em FBD:



Exemplo em ST:

```
VarSTRING1 := LEFT ('SUSI', 3);
```

## RIGHT

Fornecido pela standard.library.

Função do tipo STRING. Retorna a string inicial da direita, para uma determinada string.

Entradas:

**STR:** STRING; string a ser analisada.

**SIZE:** INT; número de caracteres a ser contado a partir da direita na string STR.

Valor de retorno:

STRING; string inicial direita.

**RIGHT (STR, SIZE):** retorna a quantidade de caracteres definida por SIZE, a partir da direita, na string STR.

Exemplo em IL:

<b>LD</b>		'SUSI'	
<b>RIGHT</b>		3	
<b>ST</b>		VarSTRING1	

O resultado é “USI”.

Exemplo em FBD:



Exemplo em ST:

```
VarSTRING1 := RIGHT ('SUSI', 3);
```

## MID

Fornecido pela standard.library.

Função do tipo STRING, retorna uma parte da string.

Entradas:

**STR:** STRING; string a ser analisada.

**LEN:** INT; comprimento da string parcial (número de caracteres).

POS: INT; posição de início para a string parcial (número de caracteres contados a partir da esquerda de STR).

Valor de retorno: STRING, string parcial.

MID (STR, LEN, POS) significa: recuperar os caracteres (definidos por LEN) da string STR iniciando com o caractere na posição POS.

Exemplo em IL:

<b>LD</b>		'SUSI'	
<b>MID</b>		2	,
		2	
<b>ST</b>		VarSTRING1	

O resultado é "US".

Exemplo em FBD:



Exemplo em ST:

```
VarSTRING1 := MID ('SUSI', 2, 2);
```

## CONCAT

Fornecido pela standard.library.

Função do tipo STRING que executa uma concatenação (combinação) de duas strings.

Entradas: STR1, STR2: STRING; strings a serem concatenadas.

Valor de retorno: STRING, string concatenada.

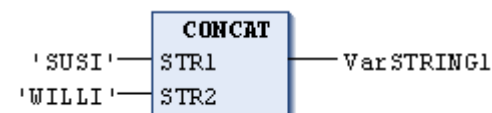
CONCAT(STR1,STR2) significa combinar STR1 e STR2 em uma string única STR1STR2.

Exemplo em IL:

<b>LD</b>		'SUSI'	
<b>CONCAT</b>		'WILLI'	
<b>ST</b>		VarSTRING1	

O resultado será "SUSIWILLI".

Exemplo em FBD:



Exemplo em ST:

```
VarSTRING1 := CONCAT ('SUSI', 'WILLI');
```

## INSERT

Fornecido pela standard.library.

Esta função, do tipo STRING, insere uma string em outra, em um ponto determinado.

Entradas:

STR1: STRING; string na qual STR2 deve ser inserida.



STR2: STRING; string a ser inserida em STR1.

POS: INT; posição em STR1 onde STR2 deve ser inserida, número de caracteres, contados a partir da esquerda.

Valor de retorno: STRING, string resultante.

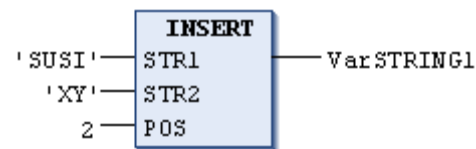
INSERT (STR1, STR2, POS) significa: inserir STR2 em STR1 após a posição POS.

Exemplo em IL:

<b>LD</b>		'SUSI'	
<b>INSERT</b>		'XY'	,
		2	
<b>ST</b>		VarSTRING1	

O resultado é 'SUXYSI'.

Exemplo em FBD:



Exemplo em ST:

```
VarSTRING1 := INSERT ('SUSI', 'XY', 2);
```

## DELETE

Fornecido pela standard.library.

Esta função, do tipo STRING, exclui uma parte de uma string maior em uma determinada posição.

Entradas:

STR: STRING; string da qual uma parte deve ser excluída.

LEN: INT; comprimento da string parcial a ser excluída (número de caracteres).

POS: INT; posição na STR onde deve iniciar a exclusão dos caracteres (LEN), contando a partir da esquerda.

Valor de retorno: STRING, a string remanescente após a exclusão.

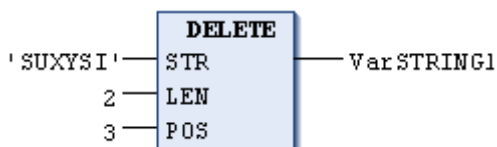
DELETE (STR, L, P) significa: excluir L caracteres da STR, começando com o caractere na posição P.

Exemplo em IL:

<b>LD</b>		'SUXYSI'	
<b>DELETE</b>		2	,
		3	
<b>ST</b>		VarSTRING1	

O resultado será "SUSI".

Exemplo em FBD:



Exemplo em ST:

```
Var1 := DELETE ('SUXYSI', 2, 3);
```

**REPLACE**

Fornecido pela standard.library.

Função do tipo STRING, que substitui uma string parcial de uma string maior por outra string.

Entradas:

STR1: STRING, string a partir da qual uma parte deve ser substituída pela string STR2.

STR2: STRING, string que deve substituir uma parte de STR1.

L: INT, comprimento da string parcial em STR1 que deve ser substituída.

P: INT, posição onde STR2 deve ser inserida no local dos L caracteres existentes.

Valor de retorno: STRING, string resultante.

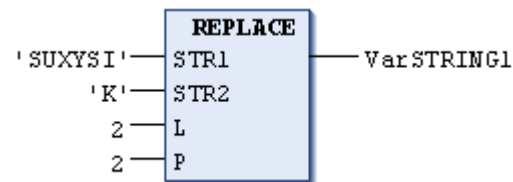
REPLACE (STR1, STR2, L, P) significa: substituir L caracteres de STR1 por STR2, começando com o caractere na posição P.

Exemplo em IL:

<b>LD</b>		'SUXYSI'	
<b>REPLACE</b>		'K'	r
		2	r
		2	
<b>ST</b>		VarSTRING1	

O resultado é "SKYSI".

Exemplo em FBD:



Exemplo em ST:

```
VarSTRING1 := REPLACE ('SUXYSI', 'K', 2, 2);
```

**FIND**

Fornecido pela standard.library.

Esta função, do tipo INT, procura a posição de uma parte da string.

Entradas:

STR1: STRING, string na qual a posição STR2 será procurada.

STR2: STRING, string cuja posição deve ser procurada em STR1.

Valor de retorno: INT, posição inicial de STR2 na STR1. Será 0 se a posição de STR2 não for encontrada em STR1.

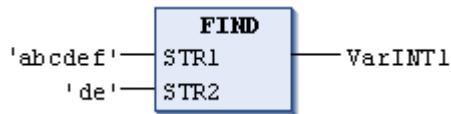
FIND (STR1, STR2) significa: encontrar a posição do primeiro caractere onde STR2 aparece na STR1 pela primeira vez. Se STR2 não for encontrada em STR1, então a função retornará 0.

Exemplo em IL:

```
LD      'abcdef'
FIND
ST      VarSTRING1
```

O resultado será “4”.

Exemplo em FBD:



Exemplo em ST:

```
arINT1 := FIND ('abcdef', 'de');
```

## Blocos Funcionais Biestáveis

### SR

Fornecido pela standard.library.

Bloco funcional que implementa biestáveis dominantes.

Entradas:

SET1: BOOL;

RESET: BOOL;

Saídas:

Q1 : BOOL;

Q1 = SR (SET1, RESET);

Q1 = (NOT RESET AND Q1) OR SET1

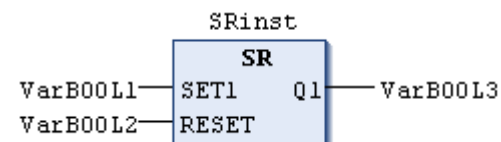
Exemplo de declaração:

```
SRInst : SR;
```

Exemplo em IL:

```
CALL      SRInst      (
                SET1:= VarBool1
                RESET:= VarBool2
            )
LD      SRInst.Q1
ST      VarBool3
```

Exemplo em FBD:



Exemplo em ST:

```
SRInst (SET1:= VarBOOL1 , RESET:=VarBOOL2 );
VarBOOL3 := SRInst.Q1 ;
```

### RS

Fornecido pela standard.library.

Bloco funcional biestável.

Entradas:

SET: BOOL;

RESET1: BOOL;

Saídas:

Q1 : BOOL.;

Q1 = RS (SET, RESET1):

Q1 = NOT RESET1 AND (Q1 OR SET)

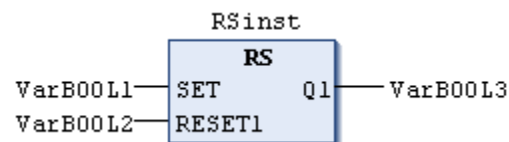
Exemplo de declaração:

```
RSInst : RS;
```

Exemplo em IL:

<b>CALL</b>		RSinst	(
	SET:=	VarBool1	,
	RESET1:=	VarBool2	)
<b>LD</b>		RSinst.Q1	
<b>ST</b>		VarBool3	

Exemplo em FBD:



Exemplo em ST:

```
RSInst(SET:= VarBOOL1 , RESET1:=VarBOOL2 );
VarBOOL3 := RSInst.Q1;
```

## Disparador

### R\_TRIG

Fornecido pela standard.library.

Bloco funcional que detecta uma borda de subida.

Entradas:

CLK: BOOL; sinal booleano de entrada a ser verificado para a borda de subida.

Saídas:

Q: BOOL; torna-se TRUE se ocorrer uma borda de subida em CLK.

A saída Q e uma variável booleana de ajuda interna M permanecerão FALSE enquanto a variável de entrada CLK retornar FALSE. Assim que CLK retornar TRUE, Q primeiro retornará TRUE e M será configurada TRUE. Isto significa que cada vez que a função for chamada, Q primeiramente será configurada para TRUE e então retornará FALSE, após uma borda de subida em CLK.

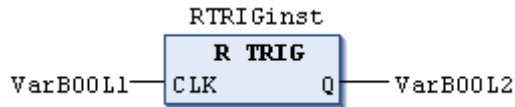
Exemplo de declaração:

```
RTRIGInst : R_TRIG;
```

Exemplo em IL:

<b>CAL</b>		RTRIGInst	(
	CLK:=	VarBoo11	)
<b>LD</b>		RTRIGInst.Q	
<b>ST</b>		VarBoo12	

Exemplo em FBD:



Exemplo em ST:

```
RTRIGInst(CLK:= VarBoo11);
VarBoo12 := RTRIGInst.Q;
```

## F\_TRIG

Fornecido pela standard.library.

Este bloco funcional detecta uma borda de descida.

Entradas: CLK: BOOL; sinal booleano de entrada a ser verificado (borda de descida).

Saídas: Q: BOOL; torna-se TRUE se ocorrer uma borda de descida em CLK.

A saída Q e uma variável booleana de ajuda interna M permanecerão FALSE enquanto a variável de entrada CLK retornar TRUE. Assim que CLK retornar FALSE, Q primeiro retornará TRUE e M será configurada para TRUE. Isto significa que cada vez que a função for chamada, Q primeiramente será configurada para TRUE e então retornará FALSE, após uma borda de descida em CLK.

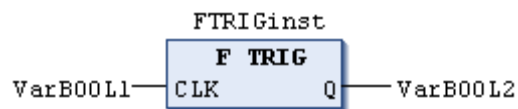
Exemplo de declaração:

```
FTRIGInst : F_TRIG;
```

Exemplo em IL:

<b>CAL</b>		FTRIGInst	(
	CLK:=	VarBoo11	)
<b>LD</b>		FTRIGInst.Q	
<b>ST</b>		VarBoo12	

Exemplo em FBD:



Exemplo em ST:

```
FTRIGInst(CLK:= VarBoo11);
VarBoo12 := FTRIGInst.Q;
```

## Contador

### CTU

Fornecido pela standard.library.

Bloco funcional que atua como um incrementador.

Entradas:

CU: BOOL; uma borda de subida nesta entrada inicia o incremento de CV.

RESET: BOOL; se TRUE, CV irá para 0.

PV: WORD; limite superior para o incremento de CV.

NOTA: O tipo de dado WORD, usado para PV no MasterTool IEC XE, não corresponde à norma IEC, a qual determina o tipo de dado INT para este.

Saídas:

Q: BOOL; torna-se TRUE assim que CV alcançar o limite determinado por PV.

CV: WORD; é incrementado até que alcance PV.

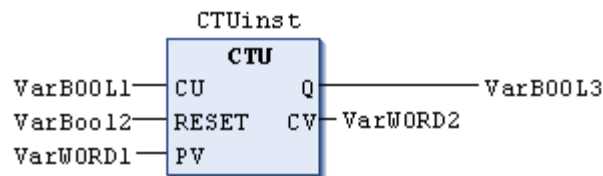
Exemplo de Declaração:

```
CTUInst : CTU;
```

Exemplo em IL:

<b>CALL</b>	CTDinst(
	CD:= VarBOOL1,
	LOAD:= VarBOOL2,
	PV:= VarWORD1,
	CV=> VarWORD2)
<b>LD</b>	CTDinst.Q
<b>ST</b>	VarBOOL3

Exemplo em FBD:



Exemplo em ST:

```
CTUInst(CU := VarBOOL1, RESET := VarBOOL2 , PV := VarWORD1);
VarBOOL3 := CTUInst.Q;
VarWORD2 := CTUInst.CV;
```

## CTD

Fornecido pela standard.library.

Bloco funcional que atua como um decrementador.

Entradas:

CD: BOOL; uma borda de subida nesta entrada inicia a decretação de CV.

LOAD: BOOL; se TRUE, CV igualará o limite superior determinado por PV.

PV: WORD; limite superior, ou seja, valor inicial para decretação de CV.

NOTA: O tipo de dado WORD, usado para PV no MasterTool IEC XE, não corresponde à norma IEC, a qual determina o tipo de dado INT para este.

Saídas:

Q: BOOL; torna-se TRUE assim que CV for 0.

CV: WORD; é decrementado em 1, iniciando com PV até que 0 seja alcançado.

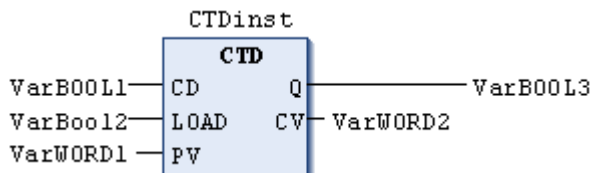
Exemplo de declaração:

```
CTDInst : CTD;
```

Exemplo em IL:

<b>CALL</b>	CTDinst(
	CD:= VarBOOL1,
	LOAD:= VarBOOL2,
	PV:= VarWORD1,
	CV=> VarWORD2)
<b>LD</b>	CTDinst.Q
<b>ST</b>	VarBOOL3

Exemplo em FBD:



Exemplo em ST:

```
CTDInst(CD := VarBOOL1, LOAD := VarBOOL2 , PV := VarWORD1);
VarBOOL3 := CTDInst.Q;
VarWORD2 := CTDInst.CV;
```

## CTUD

Fornecido pela standard.library.

Bloco funcional que atua como incrementador e decrementador.

Entradas:

CU: BOOL; se ocorrer uma borda de subida em CU, a incrementação de CV será iniciada.

CD: BOOL; se ocorrer uma borda de subida em CD, a decrementação de CV será iniciada.

RESET: BOOL; se TRUE, CV será zerado.

LOAD: BOOL; se TRUE, CV será configurado para o valor de PV.

PV: WORD; limite superior para incremento ou decremento de CV.

NOTA: O tipo de dado WORD, usado para PV no MasterTool IEC XE, não corresponde à norma IEC, a qual determina o tipo de dado INT para este.

Saídas:

QU: BOOL; retorna TRUE quando CV for incrementado em um valor  $\geq$  PV.

QD: BOOL; retorna TRUE quando CV for decrementado até 0.

CV: WORD; é incrementado ou decrementado.

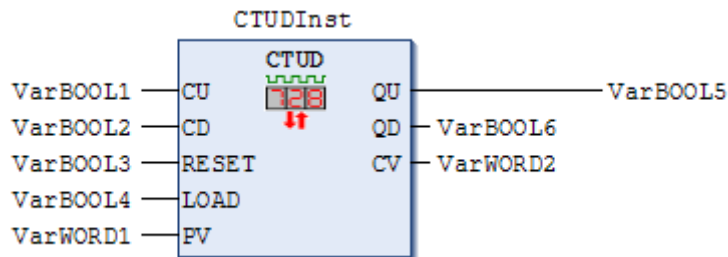
Exemplo de declaração:

```
CTUDInst : CUTD;
```

Exemplo em IL:

<b>CAL</b>	CTUDinst(
	CU:= VarBOOL1,
	RESET:= VarBOOL3,
	LOAD:= VarBOOL4,
	PV:= VarWORD1,
	QD=> VarBOOL6,
	CV=> VarWORD2)
<b>LD</b>	CTUDinst.QU
<b>ST</b>	VarBOOL5

Exemplo em FBD:



Exemplo em ST:

```
CTUDInst(CU := VarBOOL1, CD := VarBOOL2, RESET := VarBOOL3, LOAD :=
VarBOOL4, PV := VarWORD1);
VarBOOL5 := CTUDInst.QU;
VarBOOL6 := CTUDInst.QD;
VarINT1 := CTUDInst.CV;
```

## Temporizador

### TP

Fornecido pela standard.library.

Bloco funcional temporizador, funcionando como disparador. O tempo é contado até que um determinado limite seja atingido. Durante a contagem, a variável de pulso é TRUE, caso contrário, é FALSE.

Entradas:

IN: BOOL; com uma borda de subida, a contagem do tempo em ET será iniciada.

PT: TIME; limite superior do tempo.

Saídas:

Q: BOOL; TRUE enquanto o tempo estiver sendo contado em ET (pulso).

ET: TIME; estado atual do tempo.

TP(IN, PT, Q, ET):

Se IN for FALSE, Q será FALSE e ET será 0.

Assim que IN tornar-se TRUE, o tempo começará a ser contado em milissegundos em ET até que seu valor seja igual a PT (e a partir de então, permanecerá constante).

Q será TRUE quando IN for TRUE e ET for igual ou menor que PT. Caso contrário, será FALSE.

Q retorna um sinal para o período de tempo dado em PT.



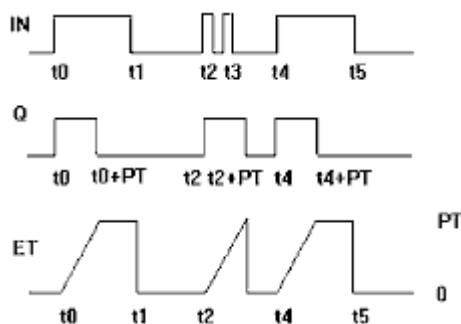


Figura 6-1. Exibição Gráfica da Sequência de Tempo TP

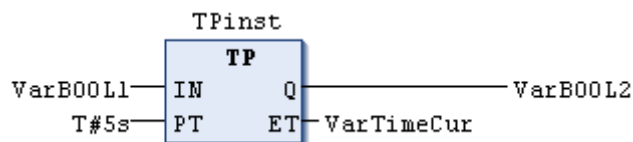
Exemplo de declaração:

```
TPInst : TP;
```

Exemplo em IL:

<b>CAL</b>	TPInst	(
	IN:= VarBOOL1	,
	PT:= T#5s	,
	ET=> VarTimeCur	)
<b>LD</b>	TPInst.Q	
<b>ST</b>	VarBOOL2	

Exemplo em FBD:



Exemplo em ST:

```
TPInst(IN := VarBOOL1, PT:= T#5s);
VarBOOL2 :=TPInst.Q;
```

## TON

Fornecido pela standard.library.

Bloco funcional de tempo que implementa um retardo na energização. Quando a entrada tornar-se TRUE, transcorrerá um tempo até que a saída torne-se TRUE.

Entradas:

IN: BOOL; borda de subida inicia a contagem ET.

PT: TIME; limite superior para contagem ET (tempo de atraso).

Saídas:

Q: BOOL; gera uma borda de subida assim que ET tiver atingido o limite superior PV (tempo de retardo esgotado).

ET: estado atual do tempo de retardo.

TON (IN, PT, Q, ET):

Se IN for FALSE, Q será FALSE e ET será 0.

Assim que IN tornar-se TRUE, o tempo começará a ser contado em milissegundos em ET até que seu valor seja igual a PT (e, a partir de então, permanecerá constante).

Q será TRUE quando IN for TRUE e ET for igual a PT. Caso contrário, será FALSE.

Assim, haverá uma borda de subida em Q quando o tempo indicado em PT (em milissegundos) tiver se esgotado.

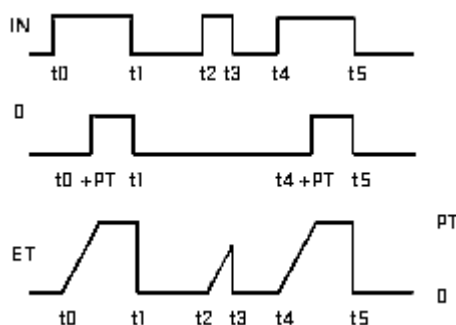


Figura 6-2. Exibição Gráfica do Comportamento de TON em Função do Tempo

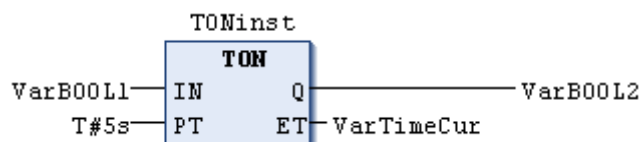
Exemplo de declaração:

```
TONInst : TON;
```

Exemplo em IL:

<b>CALL</b>	TONInst	(
	IN:= VarBOOL1	,
	PT:= T#5s	,
	ET=> VarTimeCur	)
<b>LD</b>	TONInst.Q	
<b>ST</b>	VarBOOL2	

Exemplo em FBD:



Exemplo em ST:

```
TONInst(IN := VarBOOL1, PT:= T#5s);
```

## TOF

Fornecido pela standard.library.

Bloco funcional de tempo que implementa um retardo na desenergização. Quando a entrada for alterada de TRUE para FALSE (borda de descida), transcorrerá certo tempo até que a saída vá para FALSE.

Entradas:

IN: BOOL; borda de descida inicia a contagem ET.

PT: TIME; limite superior para contagem ET (tempo de atraso).

Saídas:

Q: BOOL; gera uma borda de descida assim que ET tiver atingido o limite superior PV (tempo de retardo esgotado).

ET: estado atual do tempo de retardo.

TOF (IN, PT, Q, ET):

Se IN for TRUE, as duas saídas serão TRUE e 0, respectivamente.

Assim que IN tornar-se FALSE, o tempo começará a ser contado em ET, em milissegundos, até que seu valor seja igual a PT (e, a partir daí, permanecerá constante).

Q será FALSE quando IN for FALSE e ET for igual a PT. Caso contrário, será TRUE.

Assim, haverá uma borda de descida em Q quando o tempo indicado em PT (em milissegundos) tiver se esgotado.

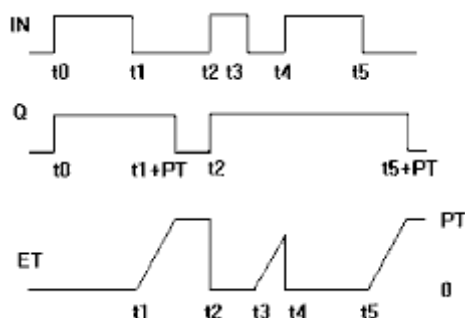


Figura 6-3. Exibição Gráfica do Comportamento de TOF em Função do Tempo

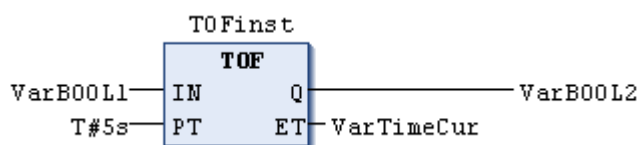
Exemplo de declaração:

```
TOFInst : TOF;
```

Exemplo em IL:

<b>CALL</b>	TOFInst	(
	IN:= VarBOOL1	,
	PT:= T#5s	,
	ET=> VarTimeCur	)
<b>LD</b>	TOFInst.Q	
<b>ST</b>	VarBOOL2	

Exemplo em FBD:



Exemplo em ST:

```
TOFInst(IN := VarBOOL1, PT:= T#5s);
VarBOOL2 :=TOFInst.Q;
```

## RTC

Fornecido pela standard.library.

O bloco funcional de tempo RunTime Clock retorna a data e a hora atual, iniciando em um determinado horário.

Entradas:

EN: BOOL; a contagem do tempo em CDT inicia em uma borda de subida.

PDT: DATE\_AND\_TIME; data e hora a partir dos quais a contagem deve iniciar.

Saídas:

Q: BOOL; é TRUE enquanto CDT estiver incrementando.

CDT: DATE\_AND\_TIME; estado atual da data e hora do contador.

VarBOOL2:= RTC (EN, PDT, Q, CDT):

Quando EN é FALSE, as variáveis de saída Q e CDT são, respectivamente, FALSE e DT#1970-01-01-00:00:00.

Assim que EN tornar-se TRUE (borda de subida) e assim permanecer, o tempo dado por PDT será definido (contado em segundos) e retornado em CDT. Assim que EN for para FALSE, CDT irá para o valor inicial DT#1970-01-01-00:00:00.

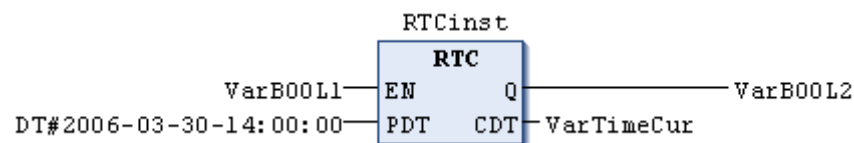
Exemplo de declaração:

```
RTCInst : RTC;
```

Exemplo em IL:

<b>CALL</b>	RTCInst	(
	EN:= VarBOOL1	,
	PDT:= DT#2006-03-30-14:00...	,
	CDT=> VarTimeCur	)
<b>LD</b>	RTCInst.Q	
<b>ST</b>	VarBOOL2	

Exemplo em FBD:



Exemplo em ST:

```
RTC (EN:=VarBOOL1, PDT:=DT#2006-03-30-14:00:00, Q=>VarBOOL2,  
CDT=>VarTimeCur);
```

## A Biblioteca UTIL.library

Esta biblioteca contém uma coleção adicional de vários blocos que podem ser usados para conversão BCD, funções bit/byte, funções matemáticas auxiliares como controlador, geradores de sinal e manipuladores de função e também para processamento de valores análogos.

### Conversão BCD

Fornecido pela util.library.

Um byte no formato BCD contém inteiros entre 0 e 99. Quatro bits são usados para cada casa decimal. A décima casa decimal é armazenada nos bits 4-7. Assim, o formato BCD é similar à representação hexadecimal, com apenas uma diferença: em um byte BCD, somente podem ser armazenados valores entre 0 e 99 e, no byte hexadecimal, de 0 a FF.

Exemplo:

Suponha que o inteiro “51” deve ser convertido para o formato BCD. Cinco em binário é “0101” e um é “0001”, o que torna o byte BCD “01010001” (e corresponde ao valor “\$51=81”).

### BCD\_TO\_INT

Fornecido pela util.library.

Esta função converte um byte em formato BCD em um valor INT.

O valor de entrada da função é do tipo BYTE e a saída é do tipo INT.

Nas conversões cujo formato não é BCD, a saída é “-1”.

Exemplos em ST:

```
i:=BCD_TO_INT(73); (* Resultado é 49 *)  
k:=BCD_TO_INT(151); (* Resultado é 97 *)
```

```
l:=BCD_TO_INT(15); (* Saída -1, pois não está no formato BCD *)
```

### *INT\_TO\_BCD*

Fornecido pela util.library.

Esta função converte um valor INTEIRO em um byte no formato BCD.

O valor de entrada da função é do tipo INT e a saída é do tipo BYTE.

O número “255” será gerado quando um valor INTEIRO, que não pode ser convertido para o formato BCD, for considerado.

Exemplos em ST:

```
i:=INT_TO_BCD(49); (* Resultado é 73 *)
k:= INT_TO_BCD(97); (* Resultado é 151 *)
l:= INT_TO_BCD(100); (* Erro! Saída: 255 *)
```

## Funções BIT/BYTE

### *Extract*

Fornecido pela util.library.

As entradas desta função são DWORD X e BYTE N. A saída é um valor BOOL com o conteúdo do enésimo bit da entrada X, onde a função inicia a contagem a partir do bit zero.

Exemplos em ST:

```
FLAG:=EXTRACT(X:=81, N:=4); (* Resultado : TRUE, porque 81 em binário é
1010001, então o quarto bit é 1 *)
FLAG:=EXTRACT(X:=33, N:=0); (* Resultado : TRUE, porque 33 em binário é
100001, então o bit '0' é 1 *)
```

### *Pack*

Fornecido pela util.library.

Esta função retorna oito bits de entrada do tipo BOOL (B0, B1, ..., B7) como um BYTE.

O bloco funcional UNPACK está relacionado a esta função. Veja mais detalhes e exemplos em **Unpack**.

### *Putbit*

Fornecido pela util.library.

As entradas desta função são DWORD X, BYTE N e um valor booleano B.

PUTBIT configura o enésimo bit da entrada X no valor B, onde a função inicia a contagem a partir do bit zero.

Exemplo em ST:

```
A:=38; (* Binário 100110 *)
B:=PUTBIT(A,4,TRUE); (* Resultado : 54 = 2#110110 *)
C:=PUTBIT(A,1,FALSE); (* Resultado : 36 = 2#100100 *)
```

### *Unpack*

Fornecido pela util.library.

UNPACK converte a entrada B do tipo BYTE em oito variáveis de saída (B0,...,B7 do tipo BOOL). É o oposto de PACK.

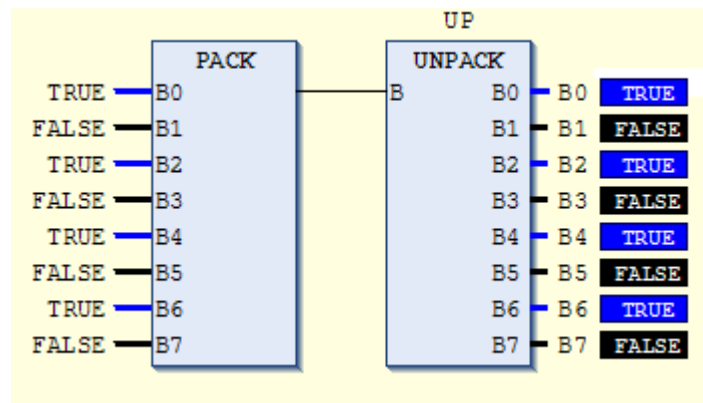


Figura 6-4. Exemplo em FBD - Saída

## Função Matemática Auxiliar

### Derivative

Fornecido pela util.library.

Este bloco funcional determina, aproximadamente, a derivação local. No uso de IN, o valor da função é devolvido como uma variável REAL.

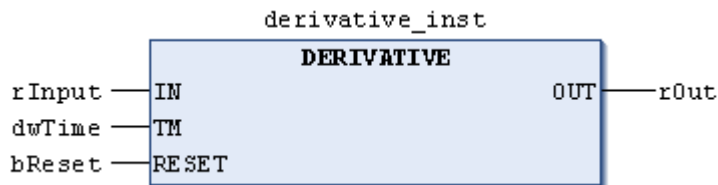
TM contém o tempo decorrido em milissegundos em uma DWORD.

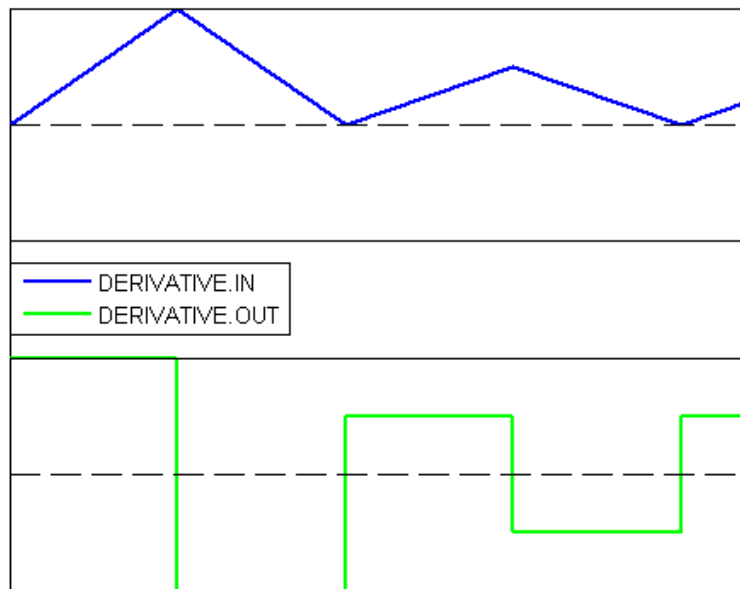
Através de uma entrada TRUE em RESET, o bloco funcional pode ser reinicializado.

A saída OUT é do tipo REAL.

Para obter o melhor resultado possível, DERIVATIVE realiza um arredondamento (usando os quatro últimos valores) para reduzir ao máximo os eventuais erros produzidos por falta de acuracidade nos parâmetros.

Exemplo em FBD:





**Figura 6-5. Comportamento do FB DERIVATIVE**

### *Integral*

Fornecido pela util.library.

Este bloco funcional determina, de forma aproximada, a integral da função.

De forma análoga à DERIVATIVE, o valor da função é devolvido com uma variável REAL através do uso de IN.

TM contém o tempo transcorrido em milissegundos em uma DWORD.

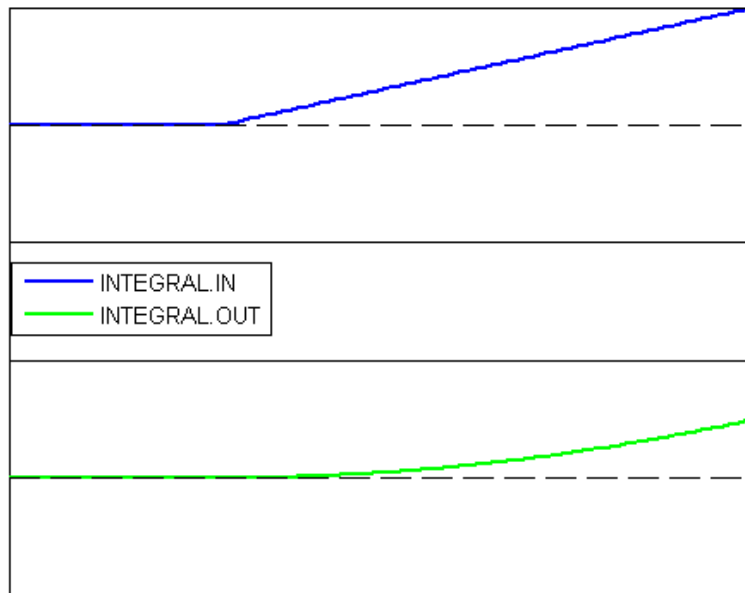
Se RESET for TRUE, o bloco funcional pode ser reiniciado.

A saída OUT é do tipo REAL.

A integral é aproximada por duas funções de passo. A média destas retorna a integral aproximada.

Exemplo em FBD:





**Figura 6-6. Comportamento do FB INTEGRAL**

### Lin\_TRAFO

Este bloco funcional (util.library) transforma um valor REAL localizado em um intervalo delimitado por um valor superior e inferior, em um valor REAL localizado em outro intervalo, igualmente definido por um limite superior e inferior. A equação  $(IN - IN\_MIN) : (IN\_MAX - IN) = (OUT - OUT\_MIN) : (OUT\_MAX - OUT)$  é a base da conversão.

Exemplo em FBD:



### Variáveis de Entrada

Variável	Tipo de dado	Descrição
IN	REAL	Valor de entrada
IN_MIN	REAL	Limite inferior do intervalo de valor da entrada
IN_MAX	REAL	Limite superior do intervalo de valor da entrada
OUT_MIN	REAL	Limite inferior do intervalo de valor da saída
OUT_MAX	REAL	Limite superior do intervalo de valor de saída

**Tabela 6-1. Variáveis de Entrada (LIN\_TRAFO)**

### Variáveis de Saída

Variável	Tipo de dado	Descrição
OUT	REAL	Valor da saída
ERROR	BOOL	Erro: TRUE, se IN_MIN = IN_MAX, ou se IN estiver fora do intervalo de valores da entrada especificada.

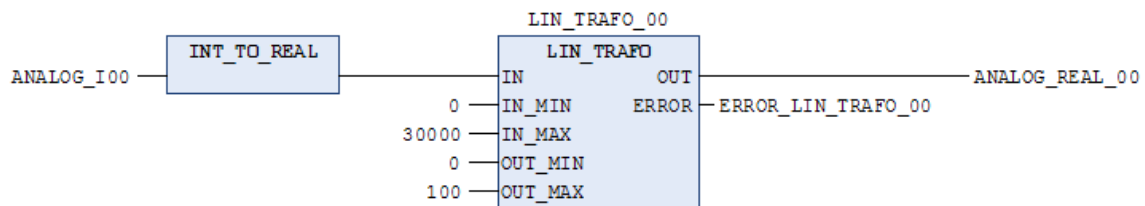
**Tabela 6-2. Variáveis de Saída (LIN\_TRAFO)**



Exemplo:

Um sensor de temperatura fornece valores em Volts numa escala de 0 a 10 V. Quando este sensor estiver conectado a uma entrada analógica de um módulo NX6000 o valor equivalente ao mínimo da escala é 0 e o valor máximo da escala é 30000. Estes valores podem ser editados na aba Parâmetros de Entrada do módulo NX6000, sendo que a faixa de 0 a 30000 corresponde aos valores padrão. Caso se queira converter os valores desta faixa para valores de temperatura em graus centígrados é possível fazer isso apenas alterando a constante de início e fim de faixa no próprio módulo. No caso em que o mínimo corresponda a 0 °C e o máximo a 100 °C, as constantes utilizadas seriam 0 para o mínimo e 100 para o máximo. Neste caso a variável convertida seria do tipo INT sem apresentar precisão menor que 1 °C.

Para fazer esta conversão sem perda de precisão é possível converter o valor lido do sensor de temperatura no formato INT e convertê-lo para um valor do tipo REAL sem perda de precisão. Neste caso, primeiro é convertido o valor de INT para REAL usando a função INT\_TO\_REAL. O resultado desta conversão deve ser passado como entrada para LIN\_TRAFO (entrada IN). Os limites de entrada são definidos pelas entradas do bloco IN\_MIN=0 e IN\_MAX=30000. O intervalo dos valores da saída (°C) é definido pelos limites OUT\_MIN=0 e OUT\_MAX=100. O valor convertido da saída será apresentado em graus centígrados (saída OUT) sem perda de precisão da leitura analógica.



**Figura 6-7. Exemplo em FBD do bloco LIN\_TRAFO**

A Figura 6-7 apresenta um exemplo de conversão de uma leitura de entrada de tensão de um módulo analógico para um valor do tipo REAL na escala de 0 a 100 °C.

Assim, uma entrada de 4,67 Vdc resultaria em um valor de entrada de 14000 correspondente a uma temperatura de 46,67 °C representada em uma variável do tipo REAL.

### STATISTICS-INT

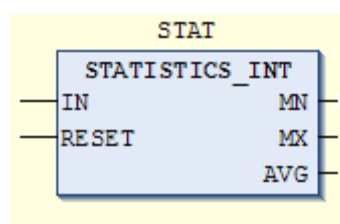
Fornecido pela util.library.

Este bloco funcional calcula alguns valores estatísticos padronizados.

A entrada IN é do tipo INT. Todos os valores são inicializados quando a entrada booleana RESET é TRUE.

A saídas MN e MX contêm os valores mínimo e máximo de IN, respectivamente. AVG descreve a média, ou seja o valor esperado de IN. A três saídas são do tipo INT.

Exemplo de STATISTICS\_INT em FBD:



### STATISTICS\_REAL

Fornecido pela util.library.

Este bloco funcional corresponde a STATISTICS\_INT, porém a entrada IN é do tipo REAL, como as saídas MN, MX, AVG.

## VARIANCE

Fornecida pela util.library.

VARIANCE calcula a variância dos valores inseridos.

A entrada IN é do tipo REAL; RESET é do tipo BOOL e a saída OUT também é do tipo REAL.

O reset de VARIANCE ocorre quando RESET=TRUE.

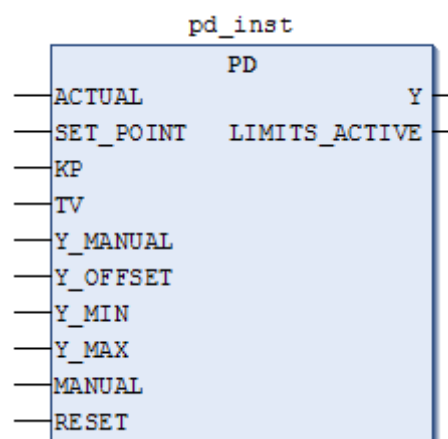
O desvio padrão pode ser facilmente calculado como a raiz quadrada de VARIANCE.

## Controllers

### PD

A biblioteca util.library fornece o bloco funcional controlador PD.

Exemplo do bloco PD em FBD:



### Entradas do Bloco Funcional

Variável	Tipo de dado	Descrição
ACTUAL	REAL	Valor atual da variável controlada.
SET_POINT	REAL	Valor desejado, variável de comando.
KP	REAL	Coefficiente de proporcionalidade, ganho de unidade da parte P.
TV	REAL	Tempo de ação derivativa, ganho de unidade da parte D em segundos, por exemplo, "0,5" para 500 milissegundos.
Y_MANUAL	REAL	Define o valor da saída Y em caso da variável MANUAL = TRUE.
Y_OFFSET	REAL	Offset para a variável Y manipulada.
Y_MIN, Y_MAX	REAL	Limites superior e inferior para a variável Y manipulada. Se Y exceder estes limites, a saída LIMITS_ACTIVE será definida para TRUE e Y será mantido neste intervalo determinado. Este controle somente funcionará se Y_MIN < Y_MAX.
MANUAL	BOOL	Se TRUE, a operação manual será ativa, ou seja, o valor manipulado será definido por Y_MANUAL.
RESET	BOOL	TRUE causa um reset no controlador. Durante a reinicialização, Y = Y_OFFSET.

**Tabela 6-3. Variáveis de Entrada (PD)**

## Saídas do Bloco Funcional

Variável	Tipo de dado	Descrição
Y	REAL	Valor manipulado, calculado pelo bloco funcional.
LIMITS_ACTIVE	BOOL	TRUE indica que Y excedeu os limites determinados (Y_MIN, Y_MAX).

Tabela 6-4. Variáveis de Saída (PD)

Y\_OFFSET, Y\_MIN e Y\_MAX são usadas para a transformação da variável manipulada em um intervalo determinado.

MANUAL pode ser usado para ligar e desligar a operação manual. RESET serve para resetar o controlador.

Na operação normal (MANUAL, RESET e LIMITS\_ACTIVE = FALSE) o controlador calcula o erro do controlador (“e”) como a diferença de SET\_POINT – ACTUAL e gera a derivada em relação ao tempo de/dt. Estes valores são armazenados internamente.

A saída, ou seja, a variável Y manipulada, é calculada da seguinte forma:  $Y = KP \times (D + TV \, dD/dt) + Y\_OFFSET$  onde  $D = SET\_POINT - ACTUAL$ .

Assim, além da parte P, também a alteração atual do erro do controlador (parte D) influencia a variável manipulada.

Adicionalmente, Y está limitado a um intervalo determinado por Y\_MIN e Y\_MAX. Se Y exceder estes limites, LIMITS\_ACTIVE se tornará TRUE. Para não haver limitação na variável manipulada, Y\_MIN e Y\_MAX devem estar configurados para 0.

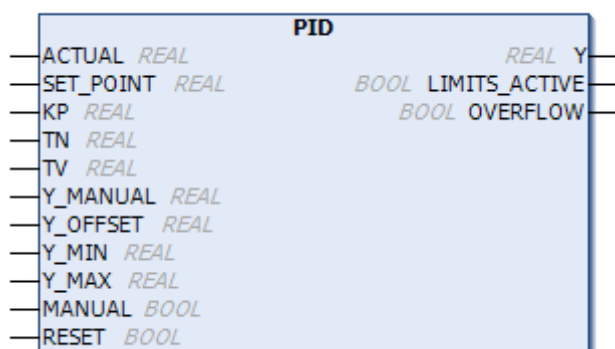
Enquanto MANUAL=TRUE, Y\_MANUAL será escrita em Y.

Um controlador P pode ser facilmente criado configurando-se TV=0.

## PID

A biblioteca util.library fornece o bloco funcional controlador PID.

Exemplo do bloco PID em FBD:



Diferentemente do controlador PD, este bloco funcional contém outra entrada, TN, do tipo REAL, para ajustar o tempo em segundos (por exemplo, “0,5” para 500 milissegundos).

## Entradas do Bloco Funcional

Variável	Tipo de dado	Descrição
ACTUAL	REAL	Valor atual da variável controlada.
SET_POINT	REAL	Valor desejado (variável de comando).
KP	REAL	Coefficiente de proporcionalidade, ganho de unidade da parte P.

<b>TN</b>	REAL	Tempo de reset, ganho de unidade recíproco da parte I. Tempo dado em segundos, por exemplo "0,5" para 500 milissegundos.
<b>TV</b>	REAL	Tempo de ação derivativa, ganho de unidade da parte D em segundos, por exemplo, "0,5" para 500 milissegundos.
<b>Y_MANUAL</b>	REAL	Define o valor da saída Y em caso da variável MANUAL = TRUE.
<b>Y_OFFSET</b>	REAL	Offset para a variável Y manipulada.
<b>Y_MIN, Y_MAX</b>	REAL	Limites superior e inferior para a variável Y manipulada. Se Y exceder estes limites, a saída LIMITS_ACTIVE será definida para TRUE e Y será mantido neste intervalo determinado. Este controle somente funcionará se Y_MIN < Y_MAX.
<b>MANUAL</b>	BOOL	Se TRUE, a operação manual será ativada, ou seja, o valor manipulado será definido por Y_MANUAL.
<b>RESET</b>	BOOL	TRUE causa um reset no controlador. Durante a reinicialização, Y = Y_OFFSET.

Tabela 6-5. Variáveis de Entrada (PID)

## Saídas do Bloco Funcional

Variável	Tipo de dado	Descrição
<b>Y</b>	REAL	Valor manipulado, calculado pelo bloco funcional.
<b>LIMITS_ACTIVE</b>	BOOL	TRUE indica que Y excedeu os limites determinados (Y_MIN, Y_MAX).
<b>OVERFLOW</b>	BOOL	TRUE indica um overflow.

Tabela 6-6. Variáveis de Saída (PID)

Y\_OFFSET, Y\_MIN e Y\_MAX servem para a transformação da variável manipulada em um intervalo determinado.

MANUAL pode ser usada para alterar para a operação manual. RESET pode ser usado para reinicializar o controlador.

Na operação normal (MANUAL = RESET = LIMITS\_ACTIVE = FALSE), o controlador calcula o erro do controlador ("e") como a diferença de SET\_POINT – ACTUAL e gera a derivada em relação ao tempo de/dt. Estes valores são armazenados internamente.

A saída, isto é, a variável Y manipulada, diferentemente do controlador PD, contém uma parte integral adicional e é calculada da seguinte forma:  $Y = KP \times (\Delta + 1/TN \int edt + TV \Delta/t) + Y\_OFFSET$ .

Assim, além da parte P, também a alteração atual do erro do controlador (parte D) e o seu histórico (parte I) influenciam a variável manipulada.

O controlador PID pode ser facilmente convertido para um controlador PI definindo-se TV=0.

Devido à parte integral adicional, um overflow pode advir da parametrização incorreta do controlador se a integral do erro D vier a divergir. Portanto, por questões de segurança, uma saída booleana chamada OVERFLOW está presente (neste caso, com o valor TRUE). Isto somente acontecerá se o sistema de controle for instável devido à parametrização incorreta. Ao mesmo tempo, o controlador será suspenso e somente será ativado novamente através da reinicialização.

NOTA: Enquanto a limitação para a variável manipulada (Y\_MIN e Y\_MAX) estiver ativa, a parte integral será adaptada (o histórico dos valores de entrada afetando automaticamente o valor de saída limitado). Se este comportamento não for o desejado, desligue a limitação no controlador PID (Y\_MIN >= Y\_MAX) e, em vez disto, aplique o operador LIMIT (norma IEC) no valor Y da saída.

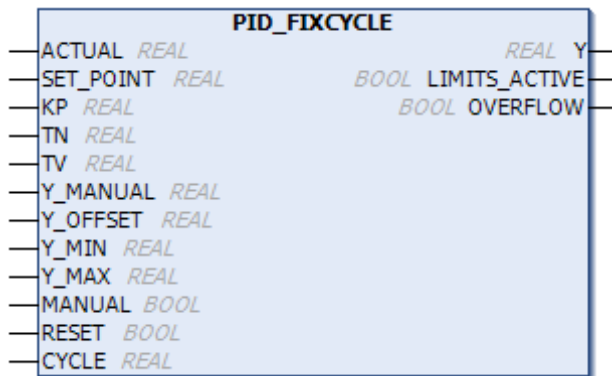
NOTA: A Altus recomenda a utilização do bloco funcional PID disponível na biblioteca NextoPID e descrita neste manual. O bloco funcional PID da biblioteca NextoPID possui parâmetros avançados para permitem um melhor ajuste do controle. As duas bibliotecas não podem ser utilizadas ao mesmo tempo.

### PID\_FIXCYCLE

Fornecido pela util.library.

Este bloco funcional corresponde ao controlador PID com a exceção de que o tempo de ciclo não é medido automaticamente por uma função interna, mas é definido por ciclo de entrada (em segundos).

Exemplo de PID\_FIXCYCLE em FBD:



### Geradores de Sinal

#### BLINK

Fornecido pela util.library.

O bloco funcional BLINK gera um sinal pulsante. A entrada consiste de ENABLE (do tipo BOOL), TIMELOW e TIMEHIGH (ambas do tipo TIME). A saída OUT é do tipo BOOL.

Se ENABLE for configurado para TRUE, BLINK passa a definir a saída para TRUE, pelo período de tempo TIMEHIGH e, posteriormente, a defini-la para FALSE, pelo período de tempo TIMELOW.

Quando ENABLE for para FALSE, a saída OUT não será alterada, ou seja, não serão mais gerados pulsos. Caso o usuário deseje que OUT seja FALSE, ele deve usar OUT AND ENABLE na saída, quando ENABLE for para FALSE (acrescentar uma caixa AND com o parâmetro ENABLE).

Exemplo de BLINK em CFC:

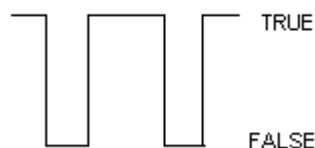
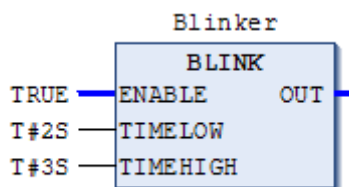


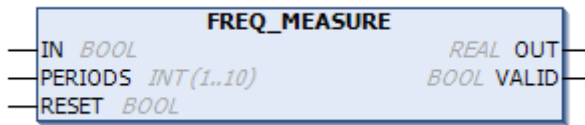
Figura 6-8. Resultado da Saída do Bloco BLINK

## FREQ\_MEASURE

Fornecido pela util.library.

Este bloco funcional mede a frequência média (Hz) de um sinal de entrada digital. É possível especificar de quantos períodos deve ser realizada a média. Um período é o tempo entre duas bordas de subida do sinal de entrada.

Exemplo de FREQ\_MEASURE em FBD:



### Variáveis de Entrada

Variável	Tipo de dado	Descrição
IN	BOOL	Sinal de entrada
PERIODS	INT	Número de períodos, ou seja, quantidade de intervalos de tempo entre as bordas de subida através dos quais a frequência média do sinal de entrada deve ser calculada. Valores possíveis: 1 a 10.
RESET	BOOL	Reset de todos os parâmetros a 0

Tabela 6-7. Variáveis de Entrada (FREQ\_MEASURE)

### Variáveis de Saída

Variável	Tipo de dado	Descrição
OUT	REAL	Frequência resultante em Hz
VALID	BOOL	FALSE até que a primeira medida tenha sido finalizada, ou se o período > 3*OUT (indicando que algo está errado com as entradas).

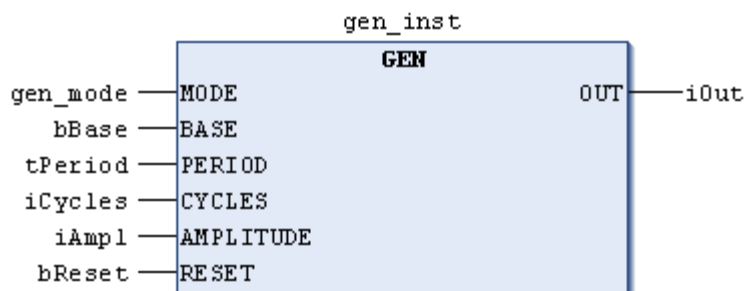
Tabela 6-8. Variáveis de Saída (FREQ\_MEASURE)

## GEN

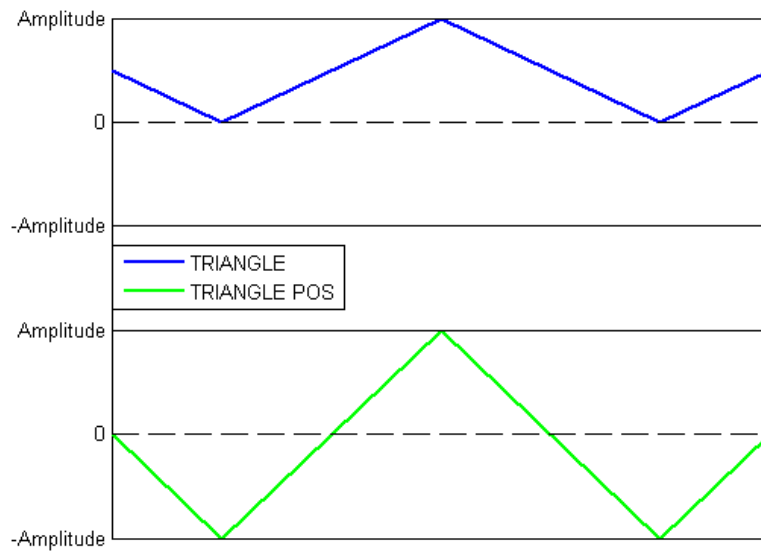
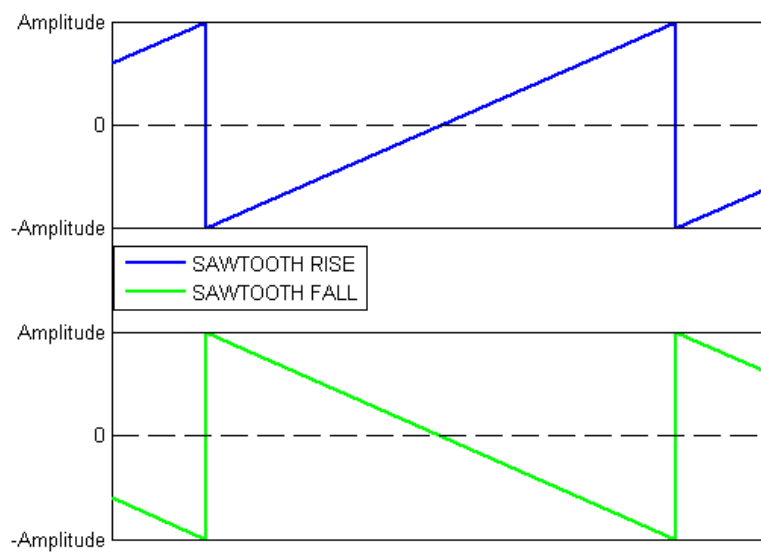
Fornecido pela util.library.

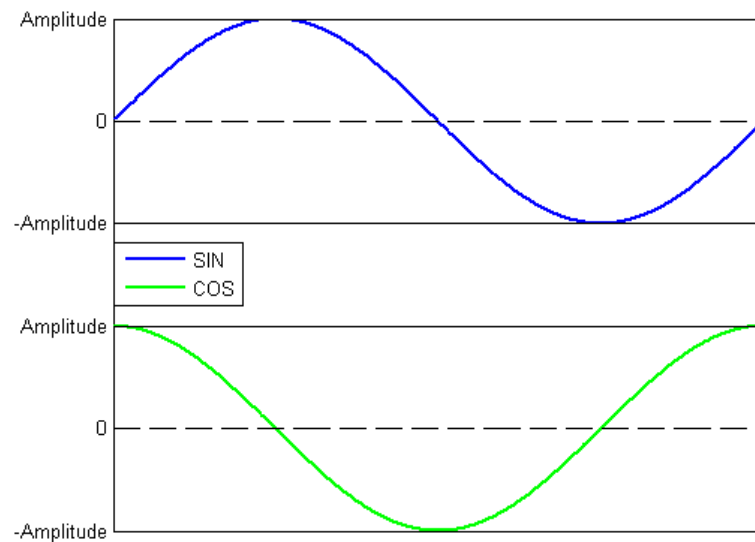
A função GEN gera funções periódicas típicas.

Exemplo de GEN em CFC:

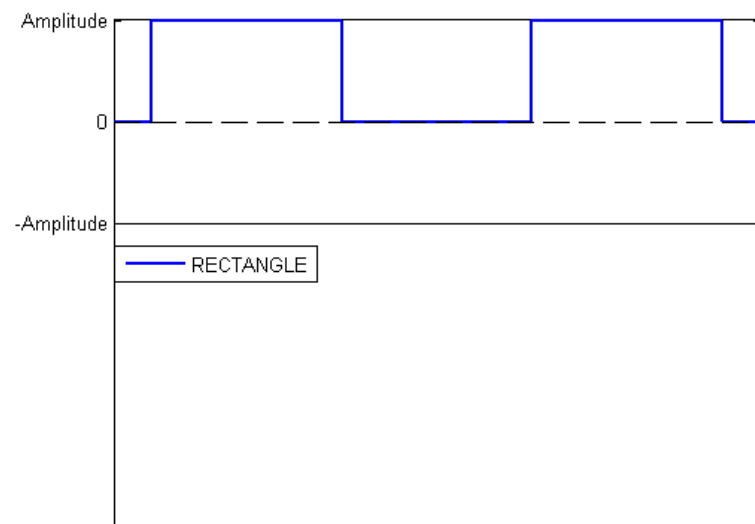


MODE descreve a função a ser gerada, onde os valores de enumeração TRIANGLE e TRIANGLE\_POS retornam duas funções triangulares, SAWTOOTH\_RISE (dente de serra ascendente), SAWTOOTH\_FALL (dente de serra descendente), RECTANGLE (sinal retangular), SINE e COSINE (seno e cosseno):

**Figura 6-9. Triângulos****Figura 6-10. Dentes de Serra**



**Figura 6-11. Seno/Cosseno**



**Figura 6-12. Retângulo**

BASE define se o período do ciclo está realmente relacionado a um tempo definido (BASE=TRUE) ou se está relacionado a um número específico de ciclos, ou seja, o número de chamadas do bloco funcional (BASE=FALSE).

PERIOD ou CYCLES define o período do ciclo correspondente.

AMPLITUDE define a amplitude da função a ser gerada.

O gerador da função vai para 0 assim que RESET=TRUE.

## Manipuladores de Função

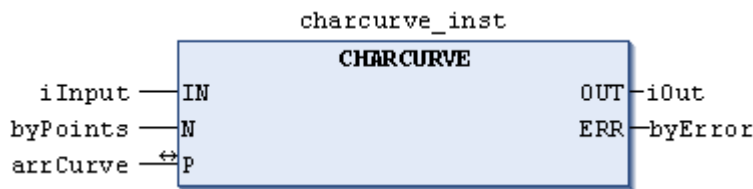
### CHARCURVE

Fornecido pela util.library.

Este bloco funcional serve para representar valores, peça por peça, em uma função linear.

Exemplo de CHARCURVE em FBD:





IN, do tipo INT, é alimentado com o valor a ser manipulado.

N, do tipo BYTE, designa o número de pontos que definem a função da apresentação.

P, do tipo ARRAY P[0..10] OF POINT, que é uma estrutura baseada em dois valores INT (X e Y), determina esta curva característica.

OUT, do tipo INT, contém o valor manipulado.

ERR, do tipo BYTE, indica um erro.

Os pontos P[0]..P[N-1] em ARRAY devem ser classificados de acordo com seus valores de X. Caso contrário, ERR receberá o valor 1. Se a entrada IN não estiver entre P[0].X e P[N-1].X, ERR=2 e OUT conterá o valor de limite correspondente P[0].Y ou P[N-1].Y.

Se N estiver fora dos valores permitidos (entre 2 e 11), então ERR=4.

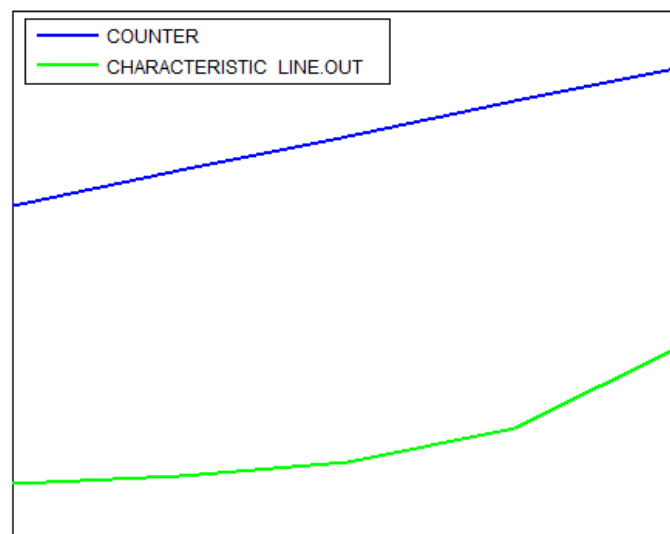
Exemplo em ST:

Primeiramente, ARRAY P deve estar definido no cabeçalho:

```
VAR
...
CHARACTERISTIC_LINE:CHARCURVE;
KL:ARRAY[0..10] OF POINT:=[ (X:=0, Y:=0), (X:=250, Y:=50),
(X:=500, Y:=150), (X:=750, Y:=400), 7 ( (X:=1000, Y:=1000) ) ];
COUNTER:INT;
...
END_VAR
```

A seguir, alimenta-se CHARCURVE com, por exemplo, um valor constantemente incrementado:

```
COUNTER:=COUNTER+10;
CHARACTERISTIC_LINE (IN:=COUNTER, N:=5, P:=KL);
```



**Figura 6-13. Ilustração das Curvas Resultantes**

## RAMP\_INT

Fornecido pela util.library.

RAMP\_INT serve para limitar a ascendência ou descendência da função de entrada.

A entrada consiste de três valores INT: IN (a entrada de função), ASCEND e DESCEND (o máximo de aumento ou diminuição por um intervalo de tempo determinado, definido por TIMEBASE do tipo TIME). Configurar RESET para TRUE faz com que RAMP\_INT seja inicializada.

A saída OUT do tipo INT contém o valor da função limitado ascendente e descendente.

Quando TIMEBASE está configurado para t#0s, ASCEND e DESCEND não estão relacionados ao intervalo de tempo e permanecem os mesmos.

Exemplo de RAMP\_INT em CFC:

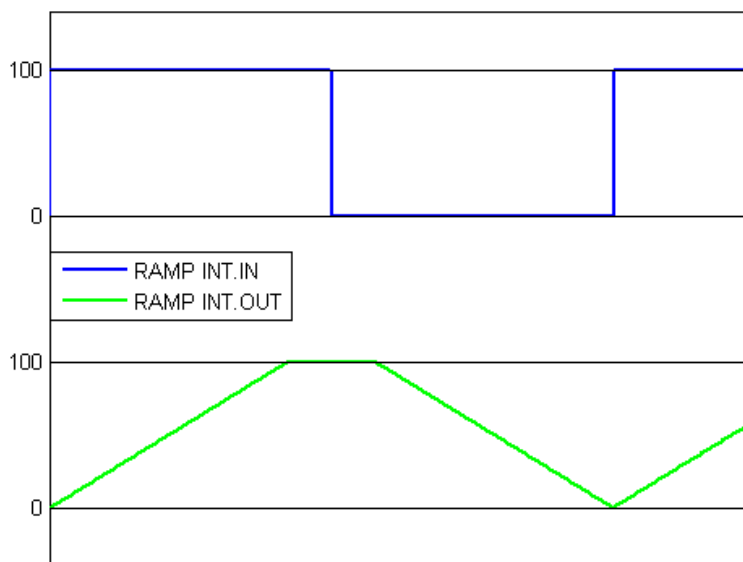
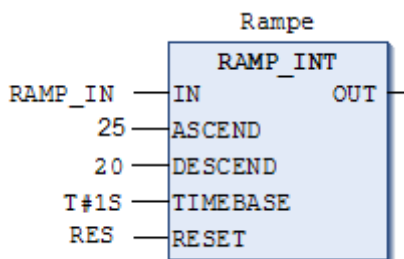


Figura 6-14. Comportamento RAMP\_INT

## RAMP\_REAL

Fornecido pela util.library.

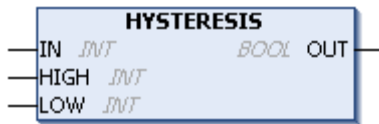
As funções RAMP\_REAL, são iguais à RAMP\_INT, com a simples diferença de que as entradas IN, ASCEND, DESCEND e a saída OUT são do tipo REAL.

## Processamento de Valores Analógicos

### HYSTERESIS

Fornecido pela util.library.

Exemplo de HYSTERESIS em FBD:



A entrada deste bloco funcional consiste de três valores INT (IN, HIGH e LOW). A saída OUT é do tipo BOOL.

Se IN estiver abaixo do valor limite LOW, OUT torna-se TRUE. Se IN exceder o limite superior HIGH, a saída será FALSE.

Se IN cair abaixo do limite LOW, OUT se tornará TRUE. A saída continuará sendo FALSE até que IN mais uma vez caia abaixo de LOW e, assim, OUT torne-se TRUE novamente.

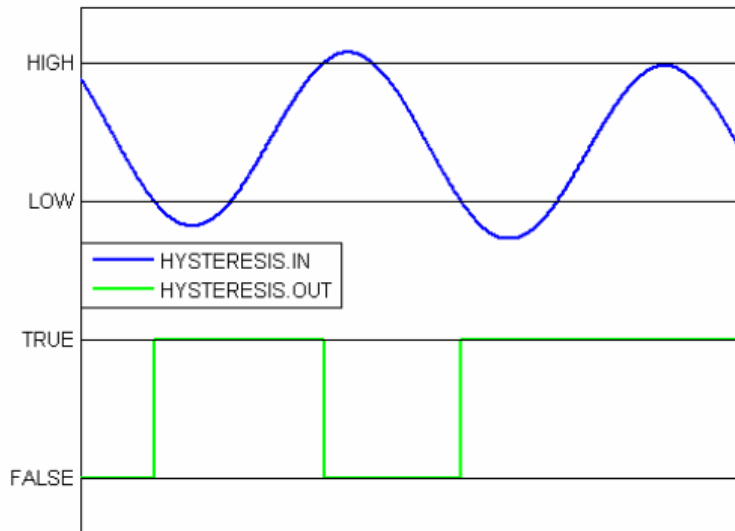


Figura 6-15. Comparação Gráfica de Hysteresis.IN e Hysteresis.OUT

## LIMITALARM

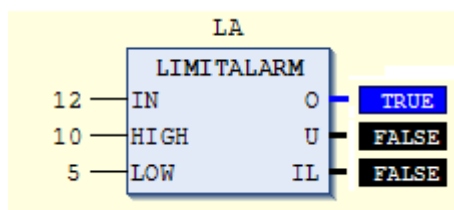
Fornecido pela util.library.

Este bloco funcional especifica se o valor da entrada está em um intervalo definido e quais limites ele violou (caso isso tenha ocorrido).

Os valores das entradas IN, HIGH e LOW são do tipo INT. As saídas O, U e IL são do tipo BOOL.

Se o limite superior HIGH for excedido por IN, O torna-se TRUE, e, quando IN estiver abaixo de LOW, U torna-se TRUE. IL é TRUE se IN estiver entre LOW e HIGH.

Exemplo de LIMITALARM em FBD:



## A Biblioteca NextoPID.library

### PID

O bloco funcional PID é utilizado para controlar um processo real. O bloco está presente na biblioteca NextoPID, a qual deve ser adicionada ao projeto.

A partir de uma variável de processo (PV) e do setpoint (SP) o bloco funcional calcula a variável manipulada (MV) para o processo controlado. Este valor é calculado periodicamente, levando em consideração os fatores proporcional, integral e derivativo programados. Trata-se de um algoritmo de controle PID tipo ISA onde o ganho proporcional é o ganho do controlador, aplicado tanto ao erro como às parcelas integral e derivativa do controlador.

O bloco funcional pode ser representado pelo diagrama básico da Figura 6-16.

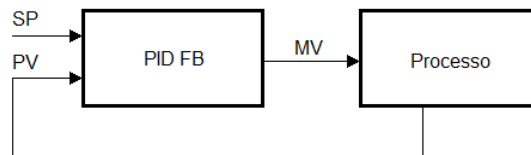


Figura 6-16. Diagrama Básico PID

A Figura 6-17 mostra o diagrama de blocos de um laço PID detalhado, conforme a execução da UCP Nexto.

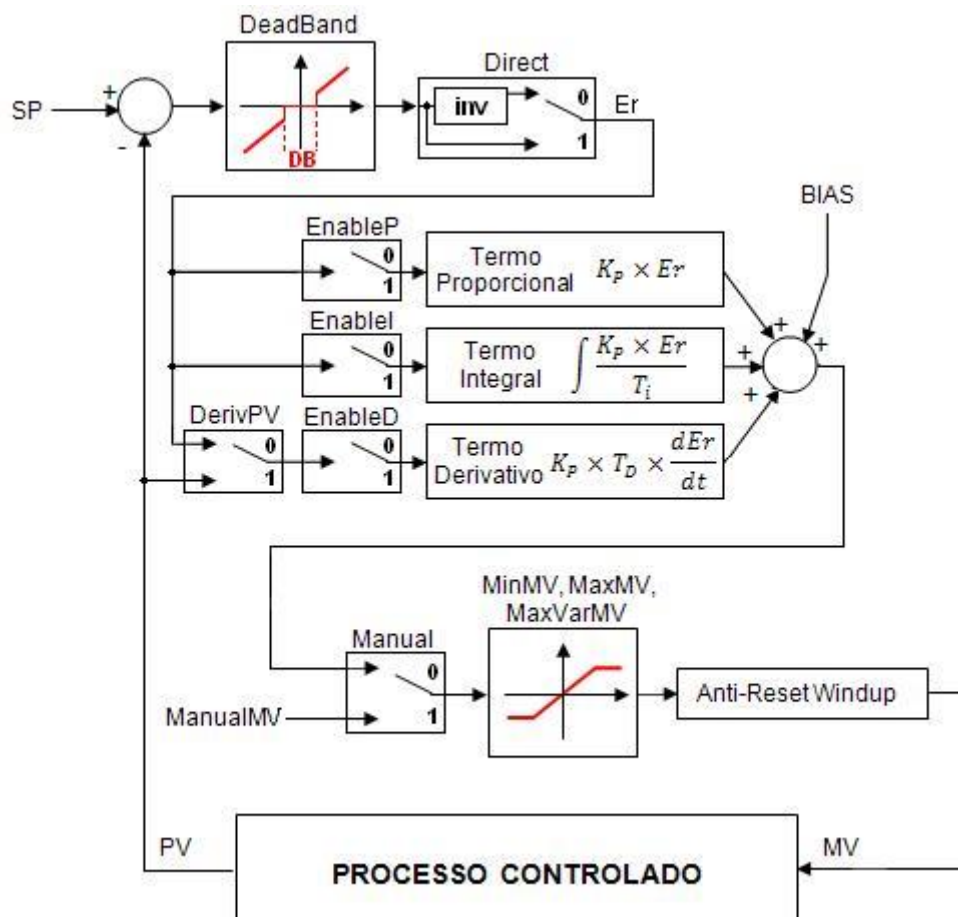
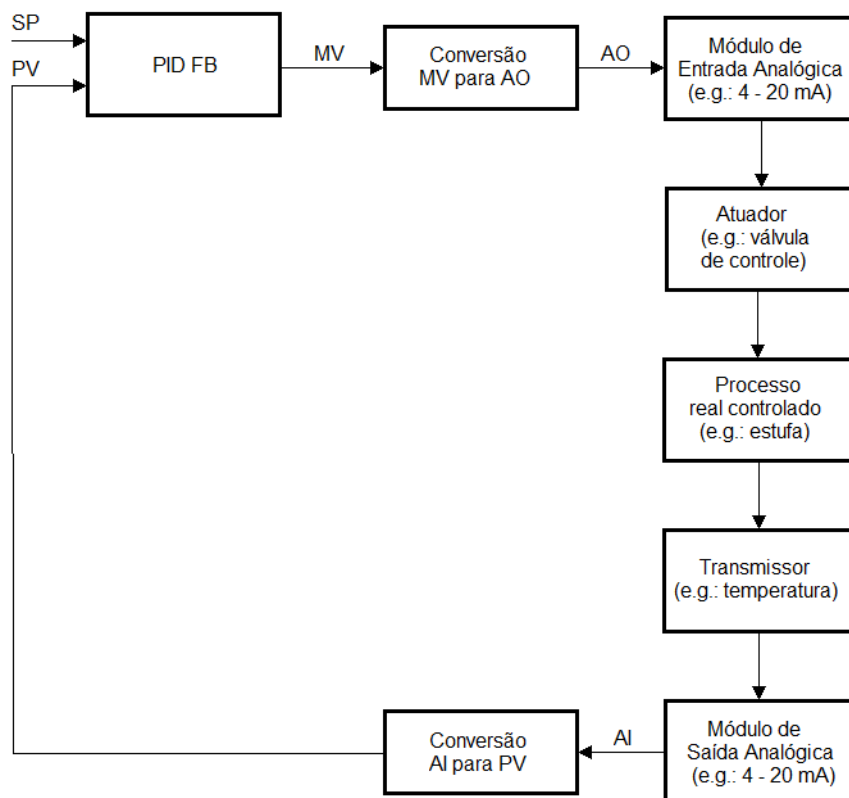


Figura 6-17. Diagrama Completo PID

A Figura 6-18 apresenta um diagrama de blocos de um exemplo de PID FB controlando um processo real. Também são apresentadas funções que auxiliam as quais o usuário deve colocar no sua aplicação (“Conversão MV para AO” e “Conversão AI para PV”).



**Figura 6-18. PID FB Controlando um Processo Real**

O diagrama apresenta apenas os principais parâmetros do bloco PID. A saída analógica (AO) é uma variável escrita no módulo de saída analógica. A entrada analógica (AI) é a variável lida de um módulo de entrada analógica.

As variáveis AI e AO normalmente são do tipo INT. Por exemplo, alguns módulos de entradas e saídas analógicas tipicamente operam na faixa de 0 .. 30000, onde 0 corresponde a 4 mA, e 30000 corresponde a 20 mA.

Por outro lado, os parâmetros MV e PV do PID são do tipo REAL. As unidades e a faixa de operação podem ser definidas da maneira mais apropriada. A seguir exemplos de conversão destes parâmetros para utilização no PID.

Considere os seguintes exemplos:

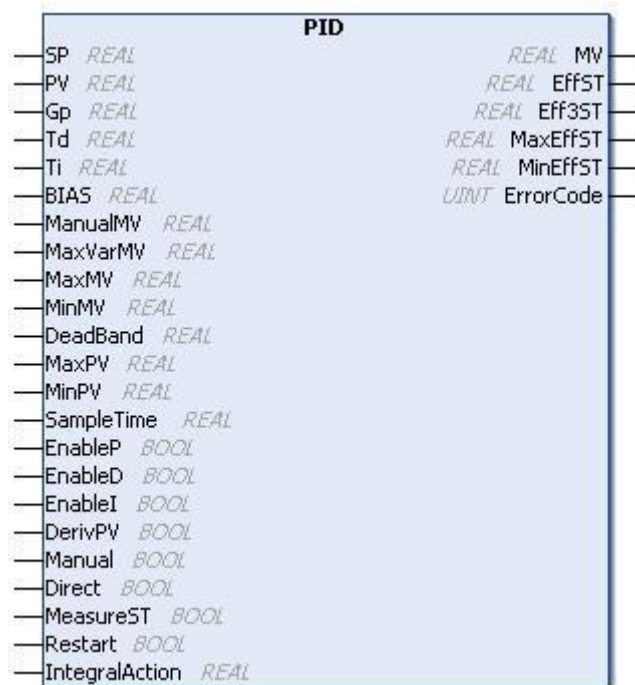
- MV com a mesma faixa de operação de AO e valor não preparado (e.g.: 0 .. 30000);
- MV em porcentagem (e.g.: 0% = válvula de controle totalmente fechada; 100% = válvula de controle totalmente aberta);

Em cada um dos exemplos, observe a necessidade de conversão que deve implementar:

- $AO := REAL\_TO\_INT (MV);$
- $AO := REAL\_TO\_INT (MV * 30000 / 100);$

NOTA: Existem alguns parâmetros do PID (descritos a seguir) que impõem valores máximo e mínimo para MV. Eles são chamados de MaxMV e MinMV respectivamente. Substituindo MV nas expressões dos exemplos anteriores por MaxMV e MinMV deve gerar valores na faixa de operação de AO (e.g.: 0 .. 30000). É necessário verificar isso para evitar problemas de overflow.

Exemplo do bloco PID em FBD:



Parâmetros de entrada	Tipo	Descrição
<b>SP</b>	REAL	Setpoint. A unidade e o intervalo devem ser os mesmos que o PV, pois as duas variáveis podem ser comparadas.
<b>PV</b>	REAL	Variável de processo. A unidade e o intervalo devem ser os mesmos que o SP, pois as duas variáveis podem ser comparadas.
<b>Gp</b>	REAL	Ganho proporcional utilizado para calcular a ação proporcional do bloco PID.
<b>Td</b>	REAL	Tempo Derivativo, em segundos, utilizado para calcular a ação derivativa do bloco PID.
<b>Ti</b>	REAL	Tempo Integral, em segundos, utilizado para calcular a ação integral do bloco PID.
<b>BIAS</b>	REAL	Compensação adicionada à variável manipulada.
<b>ManualMV</b>	REAL	Valor atribuído à variável manipulada, quando utilizado o modo manual.
<b>MaxVarMV</b>	REAL	Máxima variação da variável manipulada, entre o ciclo atual e o ciclo anterior. Caso seja zero ou negativa, o bloco PID não tem limite de variação de MV.
<b>MaxMV</b>	REAL	Máximo valor da variável manipulada. Caso o valor calculado seja maior do que o configurado, o MV será igual ao MaxMV.
<b>MinMV</b>	REAL	Mínimo valor da variável manipulada. Caso o valor calculado seja menor do que o configurado, o MV será igual ao MinMV.
<b>DeadBand</b>	REAL	Tempo morto. Mínimo valor de erro que irá causar a correção de MV em modo automático, ou seja, pequenos erros (menores que DeadBand) não causarão alterações na variável manipulada.
<b>MaxPV</b>	REAL	Máximo valor da variável de processo. Caso o valor de PV seja maior do que o configurado, o bloco PID irá parar o cálculo e será gerado um código de erro na saída.
<b>MinPV</b>	REAL	Mínimo valor da variável de processo. Caso o valor de PV seja menor do que o configurado, o bloco PID irá parar o cálculo e será gerado um código de erro na saída.

<b>SampleTime</b>	REAL	Tempo de amostragem. Define o período de chamada do bloco PID, em segundos, podendo variar de 0,001 s à 1000 s. Esse parâmetro é desconsiderado se o MeasureST for verdadeiro.
<b>EnableP</b>	BOOL	Quando verdadeiro, habilita a ação proporcional do bloco PID. Caso seja falso, a ação proporcional é zerada.
<b>EnableD</b>	BOOL	Quando verdadeiro, habilita ação derivativa do bloco PID. Caso seja falso, a ação derivativa é zerada.
<b>EnableI</b>	BOOL	Quando verdadeiro, habilita a ação integral do bloco PID. Caso seja falsa, a ação integral é zerada.
<b>DerivPV</b>	BOOL	Quando verdadeiro, a ação derivativa é calculada na variável de processo, sendo diferente de zero somente quando PV é alterado. Caso seja falso, a ação derivativa é calculada no erro, sendo dependente das variáveis SP e PV.
<b>Manual</b>	BOOL	Quando verdadeiro, habilita o modo manual. Caso seja falso, habilita o modo automático. O modo de controle do bloco PID afeta a maneira como o MV e a ação integral são calculadas.
<b>Direct</b>	BOOL	Quando verdadeiro, seleciona-se o controle direto, fazendo com que MV seja incluído na resposta para ser incluído no PV. Caso seja falso, seleciona-se o controle reverso, fazendo com que MV seja diminuído da resposta para ser incluído no PV.
<b>MeasureST</b>	BOOL	Quando verdadeiro, o tempo de amostragem é medido. Caso seja falso, o tempo de amostragem é informado pelo usuário na variável SampleTime.
<b>Restart</b>	BOOL	Quando verdadeiro, o bloco PID é reiniciado, inicializando todas as variáveis. Também pode ser utilizado para limpar as ações integral e derivativa; e os códigos de erro na saída do bloco.
<b>IntegralAction</b>	REAL	Armazena a ação integral, a qual é eliminada em estado de erro.

Tabela 6-9. Parâmetros de Entrada

Parâmetros de saída	Tipo	Descrição
<b>MV</b>	REAL	Variável manipulada.
<b>EffST</b>	REAL	Tempo efetivo de amostragem, em segundos, utilizado para os cálculos de ação derivativa e taxa limite de MV.
<b>Eff3ST</b>	REAL	Tempo efetivo de amostragem dos três últimos ciclos, em segundos, utilizado para os cálculos de ação derivativa.
<b>MaxEffST</b>	REAL	Máximo valor do tempo efetivo de amostragem, em segundos, desde a inicialização do bloco PID.
<b>MinEffST</b>	REAL	Mínimo valor do tempo efetivo de amostragem, em segundos, desde a inicialização do bloco PID.
<b>ErrorCode</b>	UINT	Código de erro exibido pelo bloco PID. Para removê-lo, basta solucionar o problema e reiniciar o bloco através da variável Restart. Abaixo, segue a descrição dos erros: 0: sem erro 1: MaxMV < MinMV 2: MaxPV < MinPV 3: PV > MaxPV 4: PV < MinPV 5: $T_i < 0,001$ s (com ação integral habilitada) 6: $T_d < 0$ s (com ação derivativa habilitada) 7: $G_p \leq 0$ 8: MaxVarMV < 0 9: DeadBand < 0 10: SampleTime < 0,001 s ou SampleTime > 1000 s (com MeasureST = falso)

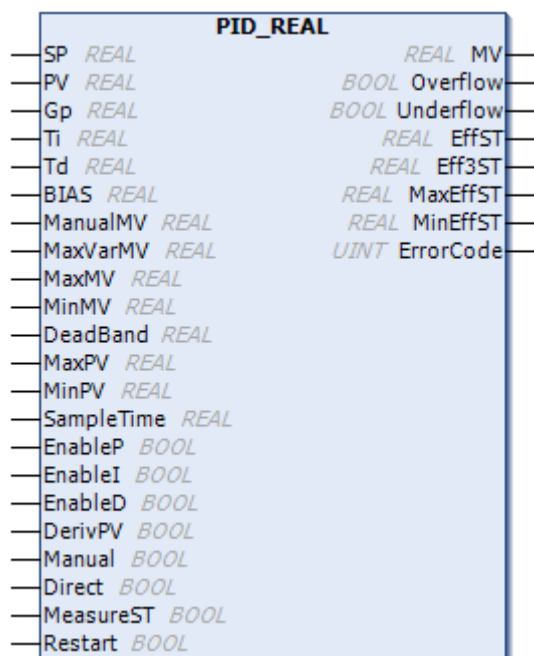
Tabela 6-10. Parâmetros de Saída

## PID\_REAL

O bloco funcional PID\_REAL implementa um algoritmo semelhante ao bloco PID desta biblioteca. Contudo este bloco testa os valores de entrada PV e SP e o valor MV para verificar se estão dentro da faixa especificada. Caso estejam fora da faixa, o cálculo do algoritmo continua sendo executado mas é gerado um código de erro na saída *ErrorCode*. Além disso se a variável MV estiver saturada, uma das saídas de *Overflow* ou *Underflow* serão ligadas.

Esta função também não possui a variável VAR\_IN\_OUT IntegralAction.

Exemplo do bloco PID\_REAL em FBD:



Parâmetros de entrada	Tipo	Descrição
<b>SP</b>	REAL	Setpoint. A unidade e o intervalo devem ser os mesmos que o PV, pois as duas variáveis podem ser comparadas.
<b>PV</b>	REAL	Variável de processo. A unidade e o intervalo devem ser os mesmos que o SP, pois as duas variáveis podem ser comparadas.
<b>Gp</b>	REAL	Ganho proporcional utilizado para calcular a ação proporcional do bloco PID.
<b>Ti</b>	REAL	Tempo Integral, em segundos, utilizado para calcular a ação integral do bloco PID.
<b>Td</b>	REAL	Tempo Derivativo, em segundos, utilizado para calcular a ação derivativa do bloco PID.
<b>BIAS</b>	REAL	Compensação adicionada à variável manipulada.
<b>ManualMV</b>	REAL	Valor atribuído à variável manipulada, quando utilizado o modo manual.
<b>MaxVarMV</b>	REAL	Máxima variação da variável manipulada, entre o ciclo atual e o ciclo anterior. Caso seja zero ou negativa, o bloco PID não tem limite de variação de MV.
<b>MaxMV</b>	REAL	Máximo valor da variável manipulada. Caso o valor calculado seja maior do que o configurado, o MV será igual ao MaxMV.
<b>MinMV</b>	REAL	Mínimo valor da variável manipulada. Caso o valor calculado seja menor do que o configurado, o MV será igual ao MinMV.
<b>DeadBand</b>	REAL	Tempo morto. Mínimo valor de erro que irá causar a correção de MV em modo automático, ou seja, pequenos erros (menores que DeadBand) não causarão alterações na variável manipulada.



<b>MaxPV</b>	REAL	Máximo valor da variável de processo. Caso o valor de PV seja maior do que o configurado, o bloco PID irá parar o cálculo e será gerado um código de erro na saída.
<b>MinPV</b>	REAL	Mínimo valor da variável de processo. Caso o valor de PV seja menor do que o configurado, o bloco PID irá parar o cálculo e será gerado um código de erro na saída.
<b>SampleTime</b>	REAL	Tempo de amostragem. Define o período de chamada do bloco PID, em segundos, podendo variar de 0,001 s à 1000 s. Esse parâmetro é desconsiderado se o MeasureST for verdadeiro.
<b>EnableP</b>	BOOL	Quando verdadeiro, habilita a ação proporcional do bloco PID. Caso seja falso, a ação proporcional é zerada.
<b>EnableD</b>	BOOL	Quando verdadeiro, habilita ação derivativa do bloco PID. Caso seja falso, a ação derivativa é zerada.
<b>EnableI</b>	BOOL	Quando verdadeiro, habilita a ação integral do bloco PID. Caso seja falsa, a ação integral é zerada.
<b>DerivPV</b>	BOOL	Quando verdadeiro, a ação derivativa é calculada na variável de processo, sendo diferente de zero somente quando PV é alterado. Caso seja falso, a ação derivativa é calculada no erro, sendo dependente das variáveis SP e PV.
<b>Manual</b>	BOOL	Quando verdadeiro, habilita o modo manual. Caso seja falso, habilita o modo automático. O modo de controle do bloco PID afeta a maneira como o MV e a ação integral são calculadas.
<b>Direct</b>	BOOL	Quando verdadeiro, seleciona-se o controle direto, fazendo com que MV seja incluído na resposta para ser incluído no PV. Caso seja falso, seleciona-se o controle reverso, fazendo com que MV seja diminuído da resposta para ser incluído no PV.
<b>MeasureST</b>	BOOL	Quando verdadeiro, o tempo de amostragem é medido. Caso seja falso, o tempo de amostragem é informado pelo usuário na variável SampleTime.
<b>Restart</b>	BOOL	Quando verdadeiro, o bloco PID é reiniciado, inicializando todas as variáveis. Também pode ser utilizado para limpar as ações integral e derivativa; e os códigos de erro na saída do bloco.

Tabela 6-11. Parâmetros de Entrada

Parâmetros de saída	Tipo	Descrição
<b>MV</b>	REAL	Variável manipulada.
<b>Overflow</b>	BOOL	Se o valor calculado pelo algoritmo do bloco funcional para a variável MV for maior que MaxMV, então MV será igual a MaxMV e esta saída será igual a TRUE.
<b>Undeflow</b>	BOOL	Se o valor calculado pelo algoritmo do bloco funcional para a variável MV for menor que MinMV, então MV será igual a MinMV e esta saída será igual a TRUE.
<b>EffST</b>	REAL	Tempo efetivo de amostragem, em segundos, utilizado para os cálculos de ação derivativa e taxa limite de MV.
<b>Eff3ST</b>	REAL	Tempo efetivo de amostragem dos três últimos ciclos, em segundos, utilizado para os cálculos de ação derivativa.
<b>MaxEffST</b>	REAL	Máximo valor do tempo efetivo de amostragem, em segundos, desde a inicialização do bloco PID.
<b>MinEffST</b>	REAL	Mínimo valor do tempo efetivo de amostragem, em segundos, desde a inicialização do bloco PID.
<b>ErrorCode</b>	UINT	Código de erro exibido pelo bloco PID. Para removê-lo, basta solucionar o problema e reiniciar o bloco através da variável Restart. Abaixo, segue a descrição dos erros: 0: sem erro 1: MaxMV < MinMV 2: MaxPV < MinPV 3: PV > MaxPV 4: PV < MinPV 5: Ti < 0,001 s (com ação integral habilitada) 6: Td < 0 s (com ação derivativa habilitada) 7: Gp ≤ 0

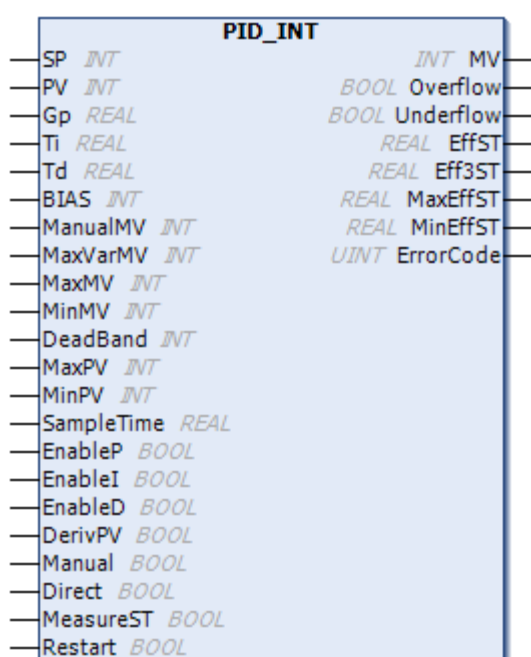
		8: MaxVarMV < 0 9: DeadBand < 0 10: SampleTime < 0,001 s ou SampleTime > 1000 s (com MeasureST = falso) 11: SP > MaxPV 12: SP < MinPV
--	--	---

Tabela 6-12. Parâmetros de Saída

## PID\_INT

O bloco funcional PID\_INT é funcionalmente idêntico ao bloco PID\_REAL. Contudo as variáveis de entrada SP, PV, BIAS, ManualMV, MaxVarMV, MaxMV, MinMV, DeadBand, MaxPV e MinPV e a variável de saída MV são do tipo INT. Este comportamento é importante pois permite declarar entradas e saídas analógicas diretamente as entradas e saídas do bloco sem ser necessárias conversões de tipos.

Exemplo do bloco PID\_INT em FBD:



Parâmetros de entrada	Tipo	Descrição
<b>SP</b>	INT	Setpoint. A unidade e o intervalo devem ser os mesmos que o PV, pois as duas variáveis podem ser comparadas.
<b>PV</b>	INT	Variável de processo. A unidade e o intervalo devem ser os mesmos que o SP, pois as duas variáveis podem ser comparadas.
<b>Gp</b>	INT	Ganho proporcional utilizado para calcular a ação proporcional do bloco PID.
<b>Ti</b>	REAL	Tempo Integral, em segundos, utilizado para calcular a ação integral do bloco PID.
<b>Td</b>	REAL	Tempo Derivativo, em segundos, utilizado para calcular a ação derivativa do bloco PID.
<b>BIAS</b>	INT	Compensação adicionada à variável manipulada.
<b>ManualMV</b>	INT	Valor atribuído à variável manipulada, quando utilizado o modo manual.
<b>MaxVarMV</b>	INT	Máxima variação da variável manipulada, entre o ciclo atual e o ciclo anterior. Caso seja zero ou negativa, o bloco PID não tem limite de variação de MV.
<b>MaxMV</b>	INT	Máximo valor da variável manipulada.

		Caso o valor calculado seja maior do que o configurado, o MV será igual ao MaxMV.
<b>MinMV</b>	INT	Mínimo valor da variável manipulada. Caso o valor calculado seja menor do que o configurado, o MV será igual ao MinMV.
<b>DeadBand</b>	INT	Tempo morto. Mínimo valor de erro que irá causar a correção de MV em modo automático, ou seja, pequenos erros (menores que DeadBand) não causarão alterações na variável manipulada.
<b>MaxPV</b>	INT	Máximo valor da variável de processo. Caso o valor de PV seja maior do que o configurado, o bloco PID irá parar o cálculo e será gerado um código de erro na saída.
<b>MinPV</b>	INT	Mínimo valor da variável de processo. Caso o valor de PV seja menor do que o configurado, o bloco PID irá parar o cálculo e será gerado um código de erro na saída.
<b>SampleTime</b>	REAL	Tempo de amostragem. Define o período de chamada do bloco PID, em segundos, podendo variar de 0,001 s à 1000 s. Esse parâmetro é desconsiderado se o MeasureST for verdadeiro.
<b>EnableP</b>	BOOL	Quando verdadeiro, habilita a ação proporcional do bloco PID. Caso seja falso, a ação proporcional é zerada.
<b>EnableD</b>	BOOL	Quando verdadeiro, habilita ação derivativa do bloco PID. Caso seja falso, a ação derivativa é zerada.
<b>EnableI</b>	BOOL	Quando verdadeiro, habilita a ação integral do bloco PID. Caso seja falsa, a ação integral é zerada.
<b>DerivPV</b>	BOOL	Quando verdadeiro, a ação derivativa é calculada na variável de processo, sendo diferente de zero somente quando PV é alterado. Caso seja falso, a ação derivativa é calculada no erro, sendo dependente das variáveis SP e PV.
<b>Manual</b>	BOOL	Quando verdadeiro, habilita o modo manual. Caso seja falso, habilita o modo automático. O modo de controle do bloco PID afeta a maneira como o MV e a ação integral são calculadas.
<b>Direct</b>	BOOL	Quando verdadeiro, seleciona-se o controle direto, fazendo com que MV seja incluído na resposta para ser incluído no PV. Caso seja falso, seleciona-se o controle reverso, fazendo com que MV seja diminuído da resposta para ser incluído no PV.
<b>MeasureST</b>	BOOL	Quando verdadeiro, o tempo de amostragem é medido. Caso seja falso, o tempo de amostragem é informado pelo usuário na variável SampleTime.
<b>Restart</b>	BOOL	Quando verdadeiro, o bloco PID é reiniciado, inicializando todas as variáveis. Também pode ser utilizado para limpar as ações integral e derivativa; e os códigos de erro na saída do bloco.

Tabela 6-13. Parâmetros de Entrada

Parâmetros de saída	Tipo	Descrição
<b>MV</b>	INT	Variável manipulada.
<b>Overflow</b>	BOOL	Se o valor calculado pelo algoritmo do bloco funcional para a variável MV for maior que MaxMV, então MV será igual a MaxMV e esta saída será igual a TRUE.
<b>Undeflow</b>	BOOL	Se o valor calculado pelo algoritmo do bloco funcional para a variável MV for menor que MinMV, então MV será igual a MinMV e esta saída será igual a TRUE.
<b>EffST</b>	REAL	Tempo efetivo de amostragem, em segundos, utilizado para os cálculos de ação derivativa e taxa limite de MV.
<b>Eff3ST</b>	REAL	Tempo efetivo de amostragem dos três últimos ciclos, em segundos, utilizado para os cálculos de ação derivativa.
<b>MaxEffST</b>	REAL	Máximo valor do tempo efetivo de amostragem, em segundos, desde a inicialização do bloco PID.
<b>MinEffST</b>	REAL	Mínimo valor do tempo efetivo de amostragem, em segundos, desde a inicialização do bloco PID.
<b>ErrorCode</b>	UINT	Código de erro exibido pelo bloco PID. Para removê-lo, basta solucionar o problema e reiniciar o bloco através da variável

	Restart. Abaixo, segue a descrição dos erros: 0: sem erro 1: MaxMV < MinMV 2: MaxPV < MinPV 3: PV > MaxPV 4: PV < MinPV 5: $T_i < 0,001$ s (com ação integral habilitada) 6: $T_d < 0$ s (com ação derivativa habilitada) 7: $G_p \leq 0$ 8: MaxVarMV < 0 9: DeadBand < 0 10: SampleTime < 0,001 s ou SampleTime > 1000 s (com MeasureST = falso) 11: SP > MaxPV 12: SP < MinPV
--	--

Tabela 6-14. Parâmetros de Saída

### Notas de Aplicação

#### Seleção do Tempo de Amostragem

A eficiência do controlador digital está diretamente relacionada com o intervalo de amostragem utilizado. A medida que este intervalo diminui, o resultado do controlador digital aproxima-se do resultado de um controlador analógico. Aconselha-se utilizar um tempo de amostragem da ordem de um décimo da constante de tempo do sistema, ou seja:  $T_A = T / 10$ , onde  $T_A$  é o tempo de amostragem utilizado e  $T$  é a constante de tempo do sistema.

Exemplo: Pode-se obter a constante de tempo de um sistema de primeira ordem a partir do seu gráfico da resposta da variável de atuação (MV) a um degrau no ponto de ajuste SP com o laço de controle aberto (PID desabilitado ou em modo manual), conforme a Figura 6-19.

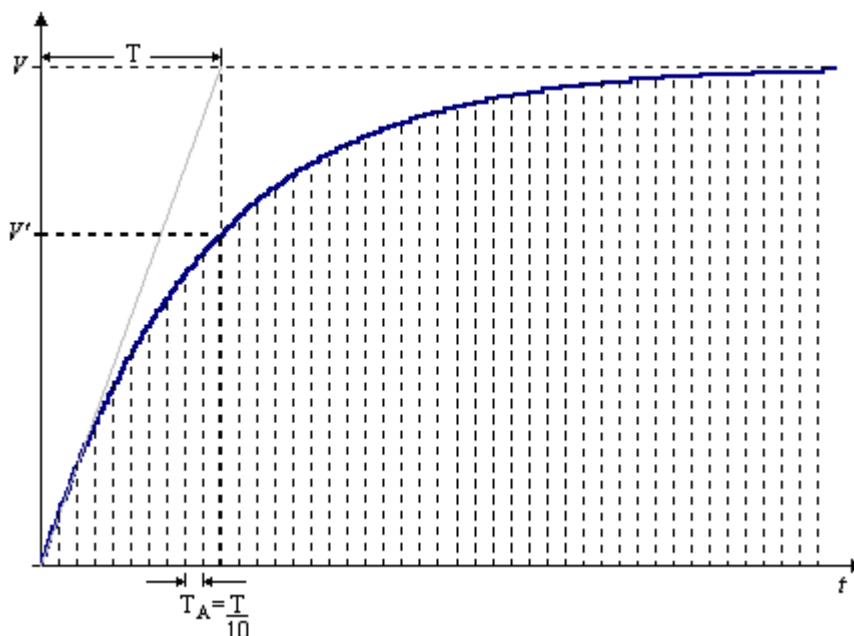


Figura 6-19. Obtenção da Constante de Tempo

A Figura 6-19 demonstra a obtenção da constante de tempo do sistema por dois modos distintos. O mais usual é tomar como constante de tempo do sistema o tempo necessário para o sistema atingir

63,212% do valor final. Outro modo é traçar a primeira derivada da curva da resposta ao degrau, a constante de tempo é aquela onde esta reta cruza o valor final da resposta do sistema.

Uma vez definida a constante de tempo, basta definir o intervalo de amostragem da ordem de um décimo deste valor.

É importante lembrar que na Série Nexto a atualização das entradas e saídas ocorre na mesma ordem de tempo de um ciclo do CP. Sempre que o tempo de ciclo do CP for maior que o tempo de amostragem aconselha-se o uso das funções `REFRESH_INPUT` e `REFRESH_OUTPUT`.

### Feedforward/Bias

Através do operando memória utilizado para feedforward/bias é possível injetar alguma variável do sistema na saída do controlador e/ou aplicar um deslocamento na mesma.

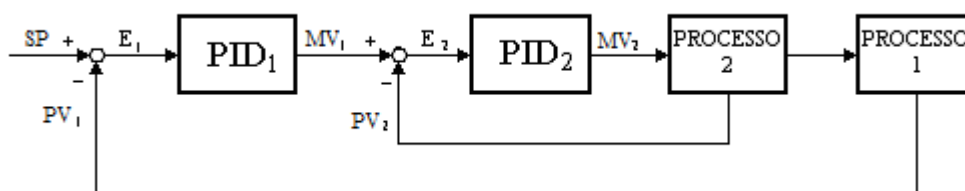
O objetivo do feedforward é medir os principais distúrbios do processo e calcular a mudança necessária na variável de atuação para compensá-los antes que estes causem alterações na variável controlada.

Pode-se citar como exemplo, um sistema onde a variável a ser controlada é a temperatura de uma mistura quente. Numa determinada fase do processo é necessário derramar água fria nesta mistura. Sem o feedforward, seria necessário esperar a água fria mudar o estado da mistura para então o controlador gerar a ação corretiva. Utilizando o feedforward, um valor associado à temperatura da água fria seria injetado na saída do controlador, fazendo com que este tome uma ação corretiva antes mesmo da água fria começar a alterar o estado da mistura quente, agilizando a resposta do controlador.

O bias é utilizado sempre que se deseja aplicar algum deslocamento sobre a saída do controlador.

### Controle em Cascata

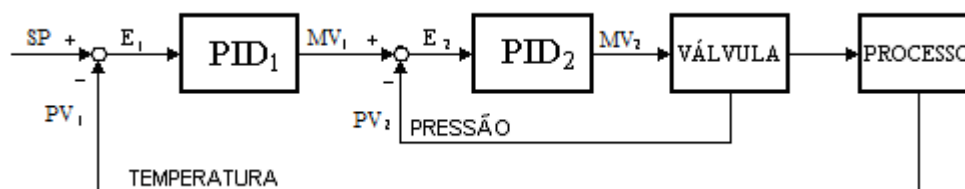
Provavelmente o controle em cascata é uma das técnicas de controle avançado mais utilizadas na prática. É composto por pelo menos duas malhas de controle. A Figura 6-20 mostra um controlador em cascata com duas malhas.



**Figura 6-20. Controle em Cascata com Duas Malhas**

A malha externa é chamada de controlador mestre e a malha interna de controlador escravo. O controlador mestre tem seu ponto de ajuste fixo e sua saída fornece o ponto de ajuste do controlador escravo (MV 1). A variável de atuação do controlador escravo (MV 2) atuará sobre o processo 2 que, por sua vez, atuará sobre o processo 1, fechando a malha do controlador mestre.

Este tipo de controlador é aplicado, por exemplo, no controle de temperatura pela injeção de vapor. Além da variação da temperatura, que deve ser controlada, o sistema está sujeito a variações de pressão na linha de vapor. Torna-se então desejável um controlador de vazão escravo atuando em função das variações de pressão e um controlador mestre para manipular a referência do escravo controlando então a temperatura do processo. Este exemplo pode ser representado graficamente conforme a Figura 6-21.



**Figura 6-21. Exemplo de Controle em Cascata Aplicado**

Caso fosse utilizado somente um controlador de temperatura atuando diretamente sobre a válvula de vapor, não haveria como compensar eventuais variações de pressão na linha de vapor.

Existem três principais vantagens no uso de controladores em cascata:

- Qualquer distúrbio que afete o controlador escravo é detectado e compensado por este controlador antes de afetar a variável controlada pelo controlador mestre.
- Aumento da controlabilidade do sistema. No caso do controle de temperatura pela injeção de vapor, a resposta do sistema é melhorada devido ao controlador de vazão aumentando a controlabilidade do laço principal.
- Não linearidades de um laço interno são manipuladas dentro deste laço e não percebidos pelo laço externo. No exemplo anterior, as variações de pressão são compensadas pelo controlador escravo e o controlador mestre enxerga apenas uma relação linear entre a válvula e a temperatura.

### Considerações Importantes

Para se utilizar controladores em cascata deve-se tomar os seguintes cuidados:

- Como o ponto de ajuste dos controladores escravos é manipulado conforme a saída dos controladores mestres, poderão ocorrer variações bruscas no erro do controlador escravo. Se os controladores escravos estiverem com a ação derivativa agindo em função do erro surgirão ações derivativas com grandes valores. Portanto aconselha-se utilizar os controladores escravos com a ação derivativa em função da variável medida.
- Controlador escravo deve ser rápido o suficiente para eliminar os distúrbios de seu laço antes que estes afetem o laço do controlador mestre.

### Sugestões para Ajustes do Controlador PID

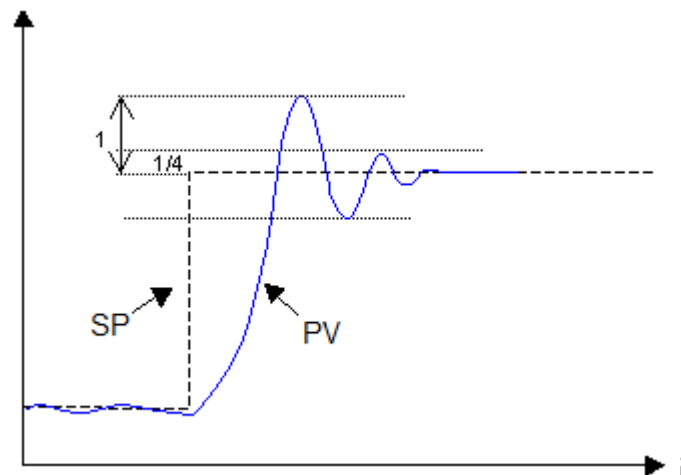
A seguir são apresentados dois métodos para a determinação das constantes do controlador PID. O primeiro método consiste na determinação das constantes em função do período de oscilação e do ganho crítico, enquanto que o segundo determina as constantes do controlador em função da constante de tempo (T), do tempo morto (T<sub>m</sub>) e do ganho estático do sistema (K). Para maiores detalhes aconselha-se a leitura da literatura referenciada.

#### ATENÇÃO:

A Altus Sistemas de Automação S.A. não se responsabiliza por eventuais danos causados por erros de configuração das constantes do controlador ou parametrização. Recomenda-se que pessoa devidamente qualificada execute esta tarefa.

### Determinação das Constantes do Controlador Através do Período e do Ganho Crítico

Este método gera uma resposta amortecida cuja taxa de amortecimento é igual a 1/4. Isto é, depois de sintonizar um laço através deste método, espera-se uma resposta como mostrada na Figura 6-22.



**Figura 6-22. Resposta Amortecida**

O ganho crítico é definido como o ganho de um controlador proporcional que gera uma oscilação de amplitude constante no sistema em malha fechada enquanto que o período crítico é o período desta oscilação. O ganho crítico é uma medida de controlabilidade do sistema, ou seja, quanto maior o ganho crítico mais fácil será o controle do sistema. O período crítico de oscilação é uma medida da velocidade de resposta do sistema em malha fechada, ou seja, quanto maior o período de oscilação mais lento será o sistema. No decorrer deste capítulo o ganho crítico será denominado como  $G_{Pc}$  e o período crítico como  $T_c$ .

É importante lembrar que ganhos ligeiramente menores que  $G_{Pc}$  geram oscilações cujo período decresce com o tempo, enquanto que ganhos maiores que  $G_{Pc}$  geram oscilações cuja amplitude cresce com o tempo. No caso de ganhos maiores que  $G_{Pc}$  é preciso ter cuidado para não tornar o sistema criticamente instável.

O processo para a determinar  $G_{Pc}$  e  $T_c$  consiste em fechar a malha com o controlador em modo automático desabilitando a ação integral e a derivativa. Os passos são os seguintes:

- Remover a ação integral e derivativa através dos respectivos parâmetros de entrada.
- Aumentar o ganho proporcional com pequenos incrementos. Depois de cada incremento inserir um pequeno distúrbio no sistema através de um pequeno degrau no ponto de ajuste (SP).
- Verificar o comportamento do sistema (PV), a amplitude de oscilação deve aumentar à medida que o ganho aumenta. O ganho crítico ( $G_{Pc}$ ) será aquele que gerar oscilações com amplitude constante (ou quase constante) conforme a Figura 7-22.
- Medir o período destas oscilações ( $T_c$ ).

Para determinar as constantes do controlador basta aplicar os valores de  $G_{Pc}$  e  $T_c$  nas fórmulas da Tabela 6-15.

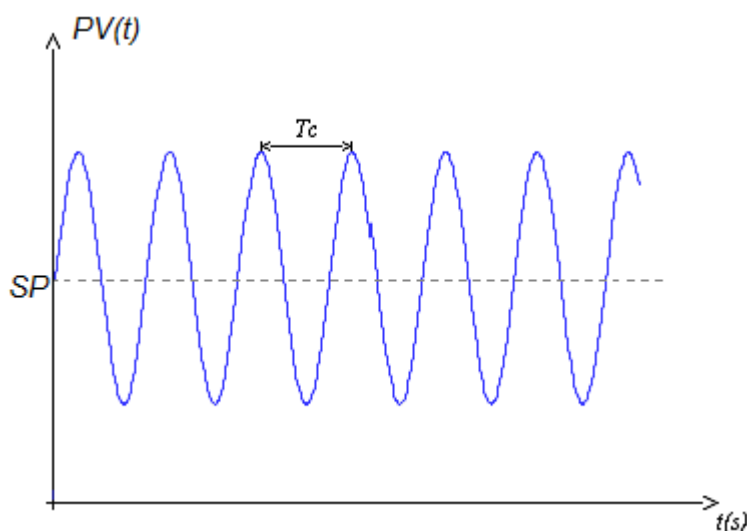


Figura 6-23. Oscilações com Amplitude Constante

Tipo de Controlador	Constantes
Proporcional (P)	$GP = 0,5.GPc$
Proporcional e Integral (PI)	$GP = 0,45.GPc$ $Ti = Tc/1,2$
Proporcional, Integral e Derivativo (PID)	$Gp = 0,75.GPc$ $Ti = Tc/1,6$ $Td = Tc / 10$

Tabela 6-15. Valores de  $GPc$  e  $Tc$

#### Determinação das Constantes do Controlador Através das Constantes do Processo

Este método se aplica bem a processos lineares, de primeira ordem (similar a um circuito RC) e com tempo morto. Na prática, muitos processos industriais se adaptam a este modelo.

O método requer, inicialmente, determinar as seguintes características do processo em laço aberto:

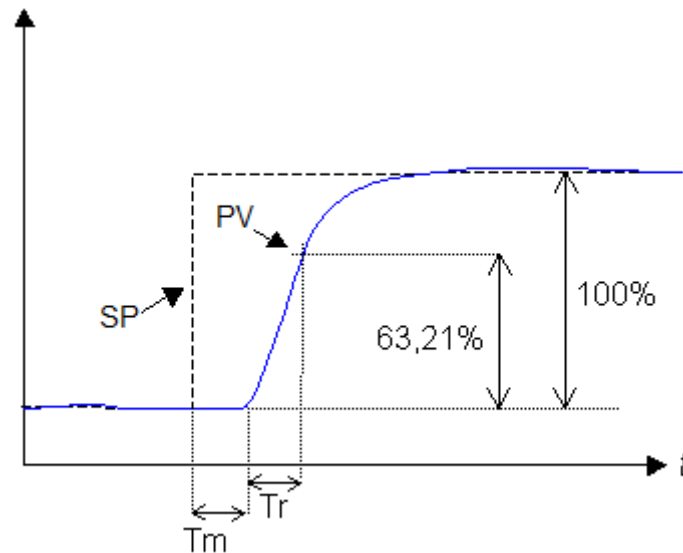
- $K$ : Ganho estático do processo. Definido como a razão entre uma variação de PV e uma variação de MV, ou seja,  $K = \Delta PV / \Delta MV$ .
- $T_m$ : Tempo morto, definido como o tempo entre o início de uma variação na saída MV ( $t_0$ ) e o início da reação do sistema.
- $T$ : Constante de tempo do sistema, definido como o tempo que a variável medida leva para excursionar 63,212% de seu valor final.

Além disso, o método requer dois parâmetros adicionais, que não são características do processo em si, e devem ser informados pelo usuário:

- $T_r$ : Tempo de resposta desejado após a sintonia do laço. Trata-se de uma característica interessante, pois através deste parâmetro o usuário pode informar um requisito de performance do laço controlado.
- $dt$ : Tempo de amostragem em segundos, isto é, o período de chamada do bloco funcional PID e atualização da entrada PV e saída MV. A constante  $dt$  simboliza um tempo morto adicional, que deve ser somado a  $T_m$ . Na prática, soma-se  $dt/2$  ao valor de  $T_m$ , pois este é o tempo morto médio inserido.

O tempo de resposta  $T_r$  pode ser comparado com uma constante de tempo do laço fechado, conforme ilustra a Figura 6-24.



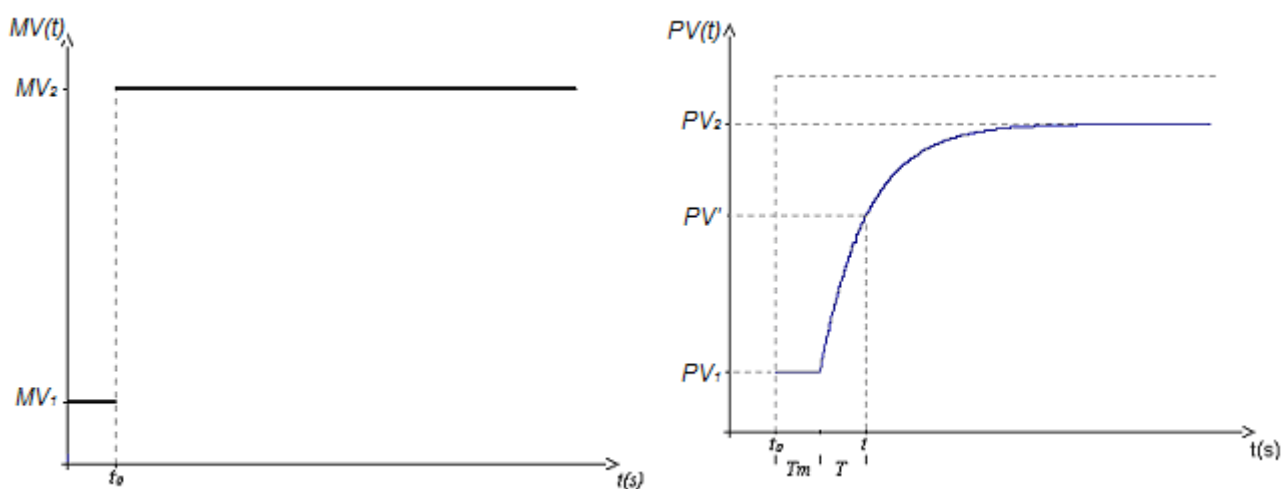


**Figura 6-24. Tr Comparado com uma Constante de Tempo**

O parâmetro  $T_r$ , na Figura 6-24, mostra o tempo de resposta desejado. Trata-se do tempo medido entre o início da resposta do sistema (após o tempo morto  $T_m$ ), e o momento em que PV atinge 63,21% de sua excursão total. Através de  $T_r$  o usuário pode especificar um requisito de performance para o laço controlado. Deve-se ter o cuidado de não especificar tempos de resposta menores que um décimo da constante de tempo do sistema, pois do contrário o sistema pode ficar instável. Quanto menor o valor de  $T_r$ , maior o ganho necessário.

A seguir, descreve-se como determinar, através de um teste de laço aberto, os demais parâmetros ( $K$ ,  $T_m$  e  $T$ ), que caracterizam o processo. Um modo simples para determinar estas constantes do processo é colocar o bloco funcional PID em modo manual, gerar um pequeno degrau em MV e plotar a resposta de PV no tempo. Para processos lentos isto pode ser feito manualmente, mas para processos rápidos aconselha-se o uso de um osciloscópio ou qualquer outro dispositivo que monitore a variação de PV. O degrau em MV deve ser grande o suficiente para causar uma variação perceptível em PV.

A Figura 6-25 representa um degrau na saída MV, aplicado no instante  $t_0$ , e a resposta de um sistema linear de primeira ordem com tempo morto.



**Figura 6-25. Degrau em MV e Resposta do Sistema ao Degrau**

Através da Figura 6-25 pode-se obter todas as constantes necessárias para a determinação dos parâmetros do controlador. O ganho estático do processo é obtido através da razão entre a variação da variável medida e a variação da variável de atuação, ou seja:

$$K = \frac{PV_2 - PV_1}{MV_2 - MV_1}$$

O tempo morto,  $T_m$ , é o tempo entre o momento de aplicação do degrau em MV ( $t_0$ ) e o início da resposta do sistema.

A constante de tempo do sistema,  $T$ , é o tempo entre o início da reação do sistema e 63,212% do valor final de PV (PV), isto é:

$$0,63212 = \frac{PV' - PV_1}{PV_2 - PV_1}$$

A partir das constantes do sistema,  $K$ ,  $T_m$  e  $T$ , pode-se obter os parâmetros do controlador utilizando as fórmulas da tabela abaixo.

Tipo de Controlador	Constantes
Proporcional, Integral e Derivativo (PID)	$GP = \frac{T}{K * (Tr + Tm + dt/2)}$ $Ti = T$ $Td = Tm/2 + dt/4$

**Tabela 6-16. Parâmetros do Controlador**

## Ganhos X Escalas

É importante lembrar que o ganho proporcional somente executará sua ação de modo correto quando tanto a entrada como a saída do sistema utilizarem as mesmas escalas. Por exemplo, um controlador proporcional com ganho unitário e entrada (PV) utilizando a faixa de 0 a 1000 somente será realmente unitário se a faixa de saída (MV) também for de 0 a 1000.

Em muitos casos as escalas de entrada e saída são diferentes. Pode-se citar como exemplo um sistema onde o cartão de entrada analógica é de 4-20 mA, onde 4 mA corresponde ao valor 0, e 20 mA corresponde ao valor 30000. E o cartão de saída analógica é de 0 V a 10 V, onde 0 V corresponde ao valor 0, e 10 V corresponde ao valor 1000. Em casos como o deste exemplo, o ajuste de escalas pode ser feito através do ganho proporcional ao invés de uma normalização dos valores de entrada ou de saída.

Uma estratégia que pode ser adotada é, inicialmente, determinar o ganho em termos percentuais (independente de escalas), sem se preocupar com o tipo de módulos de entrada e saída analógicas utilizados. Posteriormente, após determinado este ganho, deve-se executar a correção de escalas, antes de introduzir o ganho proporcional no bloco funcional PID.

A estratégia consiste em determinar o ganho proporcional do sistema utilizando a faixa percentual (0% a 100%) tanto da variável medida (PV) como do valor de atuação (MV), sem levar em consideração os valores absolutos, tanto de PV como de MV.

Isto levará à determinação de um ganho proporcional denominado GP%. Este ganho GP% não pode ser utilizado diretamente no bloco funcional PID. Antes é necessário fazer uma correção de escalas, que considere os valores absolutos destas variáveis.

### ATENÇÃO:

Na seção anterior, **Sugestões para Ajustes do Controlador PID**, são sugeridos métodos de ajuste nos quais a correção de escalas é implícita ao método, não devendo ser considerada. No capítulo seguinte, **Exemplo de Aplicação**, a correção de escalas também é desnecessária, pois utilizou-se um dos métodos abordados na seção **Sugestões para Ajustes do Controlador PID**.

A correção de escalas é ilustrada a partir de um exemplo descrito a seguir.

Considere um sistema de ar condicionado onde o módulo de entrada analógica está lendo um resistor PTC (coeficiente térmico positivo) e o módulo de saída analógica gera uma tensão de 0 a 10V para atuar sobre a válvula responsável pela circulação da água que resfria o ar insuflado.

O módulo de entrada trabalha com uma faixa de 0 a 30000, porém a faixa útil é de 6634 a 8706 com o seguinte significado:

- $EA_0 = 6634 = 0\% = 884,6 \Omega$  (corresponde a mínima temperatura que pode ser medida)
- $EA_1 = 8706 = 100\% = 1160,9 \Omega$  (corresponde a máxima temperatura que pode ser medida)

O módulo de saída utiliza a mesma faixa de 0 a 30000 sem restrições e com o seguinte significado:

- $SA_0 = 0 = 0\% = 0 \text{ V}$  (corresponde a mínima vazão de água pela válvula)
- $SA_1 = 30000 = 100\% = 10 \text{ V}$  (corresponde a máxima vazão de água pela válvula)

Supondo que o ganho GP% foi previamente determinado, o ganho GP pode ser calculado pela seguinte equação:

$$GP = GP\% * R$$

Onde:

$$R = \frac{SA_1 - SA_0}{EA_1 - EA_0}$$

Para o exemplo anterior:

$$R = \frac{3000 - 0}{8706 - 6634} = 14,478$$

Esta razão R é uma constante que, quando multiplicada pelo ganho proporcional do controlador, compensa as diferenças entre as faixas de entrada e saída sem a necessidade de uma normalização direta.

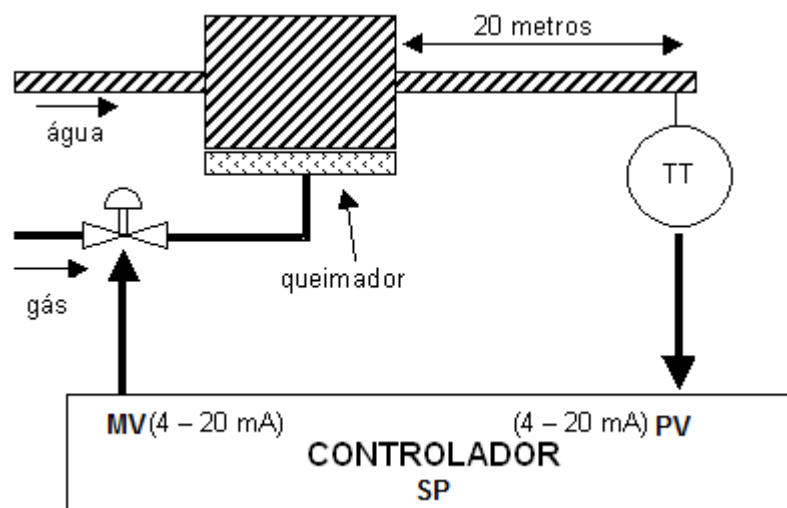
### Exemplo de Aplicação

Nesta seção, será mostrado um exemplo prático de utilização do bloco funcional PID, abrangendo diversas fases do projeto do processo e do seu sistema de controle.

### Descrição do Processo

O processo exemplo tem como objetivo o fornecimento de água aquecida, com temperatura controlada, para um consumidor. O aquecimento será feito através de um queimador de gás, sendo controlado a partir da variação de vazão de gás através de uma válvula.

A Figura 6-26 ilustra este processo.



**Figura 6-26. Exemplo de Controle de Temperatura**

Observa-se que o transmissor de temperatura (TT) fica perto do consumidor, que fica a 20 metros do ponto de aquecimento da água. Processos como este são bons exemplos de como podem ser introduzidos tempos mortos. Isto porque a água aquecida no ponto de aquecimento leva algum tempo para percorrer a distância até o ponto de medição junto do consumidor. Tempos mortos foram discutidos anteriormente (Figura 6-24).

Algumas hipóteses foram assumidas no modelamento deste processo:

- Assume-se que a água que chega ao ponto de aquecimento sobre o queimador tem temperatura fixa, de 30 °C.
- Assume-se que a vazão de água é constante.
- A seguir define-se algumas características deste processo e dos elementos utilizados:
- A água aquecida deve ter sua temperatura programável entre 50 °C e 80 °C.
- O transmissor de temperatura TT tem saída de 4 a 20 mA, e se comporta de forma linear, de tal maneira que 4 mA correspondem a 30 °C e 20 mA correspondem a 130 °C.
- Assume-se que, para aumentar em 10 °C a temperatura da água, é necessário injetar 1 m<sup>3</sup>/h de gás. Este comportamento é linear.
- A válvula de gás se fecha com 4 mA, injetando 0 m<sup>3</sup>/h de gás. Por outro lado, com 20 mA, ela injeta 8 m<sup>3</sup>/h de gás.

### Descrição dos Módulos Analógicos

Conforme pode ser visto na Figura 6-26, necessita-se de uma saída analógica 4 a 20 mA, e de uma entrada analógica de 4 a 20 mA, como interfaces entre o controlador e o processo.

Internamente ao controlador, estas faixas de 4 a 20 mA correspondem a variáveis (PV e MV). Estas faixas de valores numéricos podem variar em função dos módulos de entrada e saída analógica selecionados. Neste exemplo, assume-se o seguinte:

entrada analógica PV (0 a 30000):

PV = 0 ---> 4 mA ---> 30 °C

PV = 30000 ---> 20 mA ---> 130 °C

saída analógica MV (0 a 10000):

MV = 0 ---> 4 mA = 0 ---> 0 m<sup>3</sup>/h

MV = 10000 ---> 20 mA ---> 8 m<sup>3</sup>/h

### Ponto de Ajuste

A variável SP deve ser utilizado para programar a temperatura desejada, entre 50 °C e 80 °C.

Como esta variável deve ser comparada com PV, ela deve ter a mesma faixa numérica de PV, ou seja:

SP = 0 ---> 30 °C

SP = 30000 ---> 130 °C

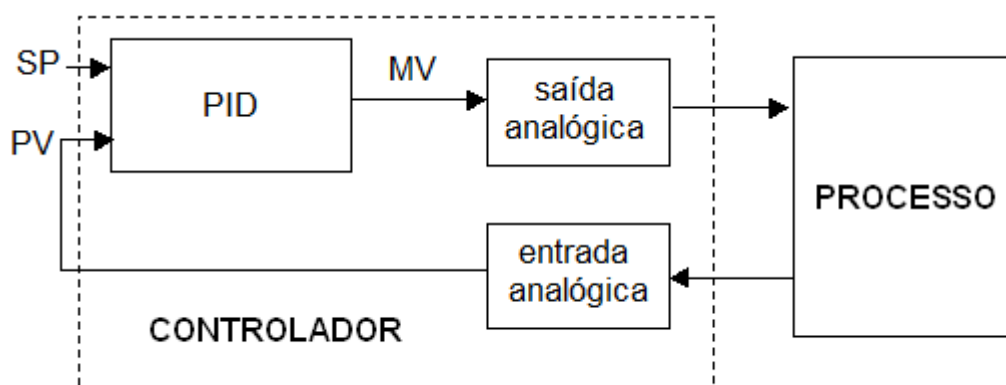
Ou para restringir a faixa entre 50 °C e 80 °C:

SP = 6000 ---> 50 °C

SP = 15000 ---> 80 °C

### Bloco diagrama Geral e Valores Limites

A Figura 6-27 mostra um bloco diagrama geral do sistema (controlador + processo), onde dentro do controlador mostra-se o bloco funcional PID. Observar que SP, PV e MV são variáveis do controlador.



**Figura 6-27. Diagrama de Blocos do Bloco Funcional PID**

SP:

mínimo = 6000 (50 °C)

máximo = 15000 (80 °C)

PV:

mínimo = 0 (30 °C)

máximo = 30000 (130 °C)

MV:

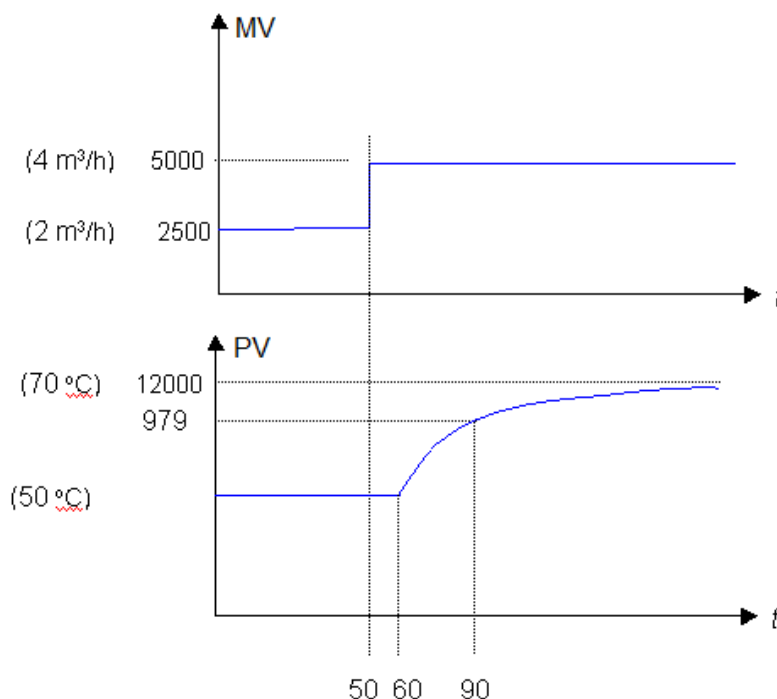
mínimo = 0 (0 m<sup>3</sup>/h)

máximo = 7500 ---> (6 m<sup>3</sup>/h)

Observa-se que no caso de MV, embora a válvula tenha capacidade de injetar 8 m<sup>3</sup>/h, deseja-se limitar esta vazão em 6 m<sup>3</sup>/h.

### Parâmetros do Processo

A Figura 6-28 mostra o resultado de um teste de malha aberta sobre o processo. Para executar este teste, utilizou-se diretamente as variáveis MV e PV, com suas unidades internas.



**Figura 6-28. Teste de Malha Aberta**

A partir da Figura 6-28 pode-se determinar os 3 parâmetros básicos, conforme explicado anteriormente na seção **Notas de Aplicação**.

$T_m = 10$  segundos (tempo morto, visto que o degrau foi aplicado em  $t = 50$  s e a resposta iniciou em  $t = 60$  s).

$T = 30$  segundos (constante de tempo, visto que a resposta iniciou em  $t = 60$  s, e atingiu 63,21% da excursão em  $t = 90$  s):

$$9792 = 6000 + (12000 - 6000) * 0,6321.$$

$K = 2.4$  (ganho estático do processo)

$$2.4 = \frac{12000 - 6000}{5000 - 2500}$$

### Sintonia do Controlador

Já que foi realizado o teste de malha aberta, será utilizado o segundo método de sintonia descrito no capítulo **Notas de Aplicação**.

Para utilizar este método, além dos parâmetros do processo determinados na seção anterior ( $T_m$ ,  $T$  e  $K$ ), também é necessário que o usuário informe outros 2 parâmetros:

- $T_r$ : Ou tempo de resposta desejado. Neste exemplo, será arbitrado em 10 segundos (um terço da constante de tempo em malha aberta).
- $dt$ : Ou tempo de ciclo do bloco funcional PID. Conforme comentado anteriormente, este tempo deve ser 10 vezes menor do que a constante de tempo em malha aberta, ou ainda menor. Portanto, o valor deve ser menor que 3 segundos. Selecionou-se  $dt = 1$  segundo.

Agora, é possível aplicar as equações do método:

$$GP = T / (K * (T_r + T_m + Dt/2)) = 30 / (2.4 * (10 + 10 + 1/2)) = 0,609$$

$$T_i = T = 30 \text{ s/rep}$$

$$T_d = T_m/2 + Dt/4 = 10/2 + 1/2 = 5.25 \text{ s}$$

## LibRecipeHandler

Esta biblioteca permite manipular receitas no CP.

### ATENÇÃO:

Para o correto funcionamento desta biblioteca uma biblioteca de sistema é utilizada. Esta biblioteca é adicionada ao projeto quando o objeto Recipe Manager for adicionado. Caso ele não seja adicionado será exibida uma mensagem de erro de compilação se a LibRecipeHandler for utilizada.

## WriteRecipe

Esta função permite escrever valores de uma receita carregada dentro de um objeto RecipeDefinition para as variáveis da aplicação que está sendo executada em um CP. Os parâmetros de entrada desta função estão descritos na Tabela 6-17.

Parâmetros de entrada	Tipo	Descrição
sRecipeDefinitionName	STRING	STRING com o nome do objeto Recipe Definition onde está a receita a ser escrita. Deve ser observado o uso de caracteres maiúsculos e minúsculos pois o nome na string deve ser idêntico ao atribuído ao objeto.
sRecipeName	STRING	STRING com o nome da receita que será escrita e está definida dentro do objeto Recipe Definition. Deve ser observado o uso de caracteres maiúsculos e minúsculos pois o nome na string deve ser idêntico ao atribuído a receita.

**Tabela 6-17. Parâmetros de Entrada**

Quando a função é executada com o sucesso o seu retorno é zero. Quando acontece algum erro um código de erro é o retorno. O retorno da função é do tipo de DWORD, mas pode ser declarado como um enumerável chamado RECIPE\_RETURN\_VALUES. os valores possíveis de retorno são apresentados na Tabela 6-18.

Código de Erro	Valor	Descrição
ERROR_OK	16#0	Função executada corretamente.
ERROR_RECIPE_NOT_FOUND	16#4003	Não existe ou está errado o nome da receita dentro do objeto Recipe Definition. Também indica erro no formato da STRING inserida como parâmetro. Por exemplo, a STRING não pode ser nula e deve possuir no máximo 60 caracteres e não pode possuir os caracteres ' ', '.' e '/'.
ERROR_RECIPE_DEFINITON_NOT_FOUND	16#4004	Não existe ou está errado o nome do objeto Recipe Definition. Também indica erro no formato da STRING inserida como parâmetro. Por exemplo, a STRING não pode ser nula e deve possuir no máximo 60 caracteres e não pode possuir os caracteres ' ', '.' e '/'.
ERROR_NO_RECIPE_MANAGER_SET	16#4006	Não existe um objeto RecipeManager no projeto onde a função foi incluída. Este erro é informado caso os parâmetros da função sejam consistentes. Caso contrário é indicado que erro no parâmetro inconsistente.

**Tabela 6-18. Códigos de Erro da Função**

Antes de executar a escrita a função consiste os parâmetros de entrada. Se houver alguma inconsistência é indicado erro na Receita ou no Recipe Definition. Caso as STRINGS dos parâmetros sejam válidas a função tenta executar a escrita. Se a Receita ou o Recipe Definition não estiver carregado no CP pode ocorrer erro.

Exemplo de uso ST com a declaração de RECIPE\_RETURN\_VALUES como retorno da função:

```
VAR
  sName : STRING := 'Recipe001';
  sDef : STRING := 'Recipes';
```

```
    mRET :RECIPE_RETURN_VALUES;  
    boolStartProcess : BOOL;  
END_VAR  
mRET := WriteRecipe(SDef,SName);  
IF mRET <> RECIPE_RETURN_VALUES.ERROR_OK THEN  
    boolStartProcess:= FALSE;  
ELSE  
    boolStartProcess := TRUE;  
END_IF;
```



## 7. Glossário

<b>Algoritmo</b>	Sequência finita de instruções bem definidas, objetivando a resolução de problemas.
<b>Árvore</b>	Estrutura de dados para configuração do hardware.
<b>Barramento</b>	Conjunto de módulos de E/S interligados a uma UCP ou cabeça de rede de campo.
<b>Bit</b>	Unidade básica de informação, cujo estado é 0 (falso) ou 1 (verdadeiro).
<b>Breakpoint</b>	Ponto de parada no aplicativo para depuração.
<b>Broadcast</b>	Disseminação simultânea de informação a todos os nós interligados a uma rede de comunicação.
<b>Byte</b>	Unidade de informação composta por oito bits.
<b>CAN</b>	Protocolo de comunicação usado largamente em redes automotivas.
<b>Controlador programável</b>	Também chamado de CP. Equipamento que realiza controle sob o comando de um programa aplicativo. É composto de uma UCP, uma fonte de alimentação e uma estrutura de E/S.
<b>CP</b>	Veja controlador programável.
<b>Diagnóstico</b>	Procedimento utilizado para detectar e isolar falhas. É também o conjunto de dados usados para tal determinação, que serve para a análise e correção de problemas.
<b>Download</b>	Carga de programa ou configuração no CP.
<b>E/S</b>	Veja entrada/saída.
<b>Entrada/saída</b>	Também chamado de E/S. Dispositivos de E/S de dados de um sistema. No caso de CPs, correspondem tipicamente a módulos digitais ou analógicos de entrada ou saída que monitoram ou acionam o dispositivo controlado.
<b>Escravo</b>	Equipamento ligado a uma rede de comunicação que só transmite dados se for solicitado por outro equipamento denominado mestre.
<b>Gateway</b>	Equipamento ou software para a conexão de duas redes de comunicação com diferentes protocolos.
<b>Hardware</b>	Equipamentos físicos usados em processamento de dados onde normalmente são executados programas (software).
<b>IEC 61131</b>	Norma genérica para operação e utilização de CPs. Antiga IEC 1131.
<b>Interface</b>	Dispositivo que adapta elétrica e/ou logicamente a transferência de sinais entre dois equipamentos.
<b>Interrupção</b>	Evento com atendimento prioritário que temporariamente suspende a execução de um programa e desvia para uma rotina de atendimento específica
<b>kbytes</b>	Unidade representativa de quantidade de memória (kB). Representa 1024 bytes.
<b>Linguagem de programação</b>	Um conjunto de regras e convenções utilizado para a elaboração de um programa.
<b>Login</b>	Ação de estabelecer um canal de comunicação com o CP.
<b>Menu</b>	Conjunto de opções disponíveis e exibidas por um programa no vídeo e que podem ser selecionadas pelo usuário a fim de ativar ou executar uma determinada tarefa.
<b>Menu de Contexto</b>	Menu dinâmico com o conteúdo de acordo com o contexto atual.
<b>Mestre</b>	Equipamento ligado a uma rede de comunicação de onde se originam solicitações de comandos para outros equipamentos da rede.
<b>Módulo (referindo-se a hardware)</b>	Elemento básico de um sistema completo que possui funções bem definidas. Normalmente é ligado ao sistema por conectores, podendo ser facilmente substituído.
<b>Operandos</b>	Elementos sobre os quais as instruções atuam. Podem representar constantes, variáveis ou um conjunto de variáveis.
<b>PC</b>	Sigla para programmable controller. É a abreviatura de controlador programável em inglês.
<b>POU</b>	<i>Program Organization Unit</i> , ou Unidade de Organização de Programa, é uma subdivisão do programa aplicativo que pode ser escrito em qualquer uma das linguagens disponíveis.
<b>Programa aplicativo</b>	É o programa carregado em um CP, que determina o funcionamento de uma máquina ou processo.
<b>Protocolo</b>	Regras de procedimentos e formatos convencionais que, mediante sinais de controle, permitem o estabelecimento de uma transmissão de dados e a recuperação de erros entre equipamentos.
<b>RAM</b>	Sigla para random access memory. É a memória onde todos os endereços podem ser acessados diretamente de forma aleatória e com a mesma velocidade. É volátil, ou seja, seu conteúdo é perdido quando o equipamento é desenergizado, a menos que se possua uma bateria para a retenção dos valores.
<b>Rede de comunicação</b>	Conjunto de equipamentos (nós) interconectados por canais de comunicação.
<b>Reset</b>	Comando para reinicializar o CP.
<b>RUN</b>	Comando para colocar o CP em modo de execução.
<b>Set</b>	Ação para atribuir o estado de nível lógico alto para uma variável booleana.
<b>Software</b>	Programas de computador, procedimentos e regras relacionadas à operação de um sistema de processamento de dados.
<b>STOP</b>	Comando para congelar o CP em seu estado atual.

<b>Token</b>	É uma marca que indica quem é o mestre do barramento no momento.
<b>Tooltip</b>	Caixa de texto com uma ajuda ou local onde pode-se entrar com a ajuda.
<b>UCP</b>	Sigla para unidade central de processamento. Controla o fluxo de informações, interpreta e executa as instruções do programa e monitora os dispositivos do sistema.
<b>UCP ativa</b>	Em um sistema redundante, a UCP ativa realiza o controle do sistema, lendo os valores dos pontos de entrada, executando o programa aplicativo e acionando os valores das saídas.
<b>Word</b>	Unidade de informação composta por 16 bits.
<b>XML</b>	Do inglês, Extensible Markup Language, é um padrão para gerar linguagens de marcação.
<b>Zoom</b>	No contexto da janela de função do teclado, é utilizado para a troca de telas.